Computer Science

First examinations 2006

DIPLOMA PROGRAMME

COMPUTER SCIENCE

First examinations 2006

International Baccalaureate Organization

Buenos Aires **Cardiff** **Geneva** **New York** **Singapore**

Diploma Programme
Computer science

International Baccalaureate Organization, Geneva, CH-1218, Switzerland

First published in April 2004

by the International Baccalaureate Organization
Peterson House, Malthouse Avenue, Cardiff Gate
Cardiff, Wales GB CF23 8GL
UNITED KINGDOM

Tel: + 44 29 2054 7777
Fax: + 44 29 2054 7778
Web site: www.ibo.org

Glossary of computer science terms

IBO merchandise and publications in its official and working languages can be purchased through the online catalogue at www.ibo.org, found by selecting *Publications* from the shortcuts box. General ordering queries should be directed to the sales department in Cardiff.

Tel: +44 29 2054 7746
Fax: +44 29 2054 7779
E-mail: sales@ibo.org

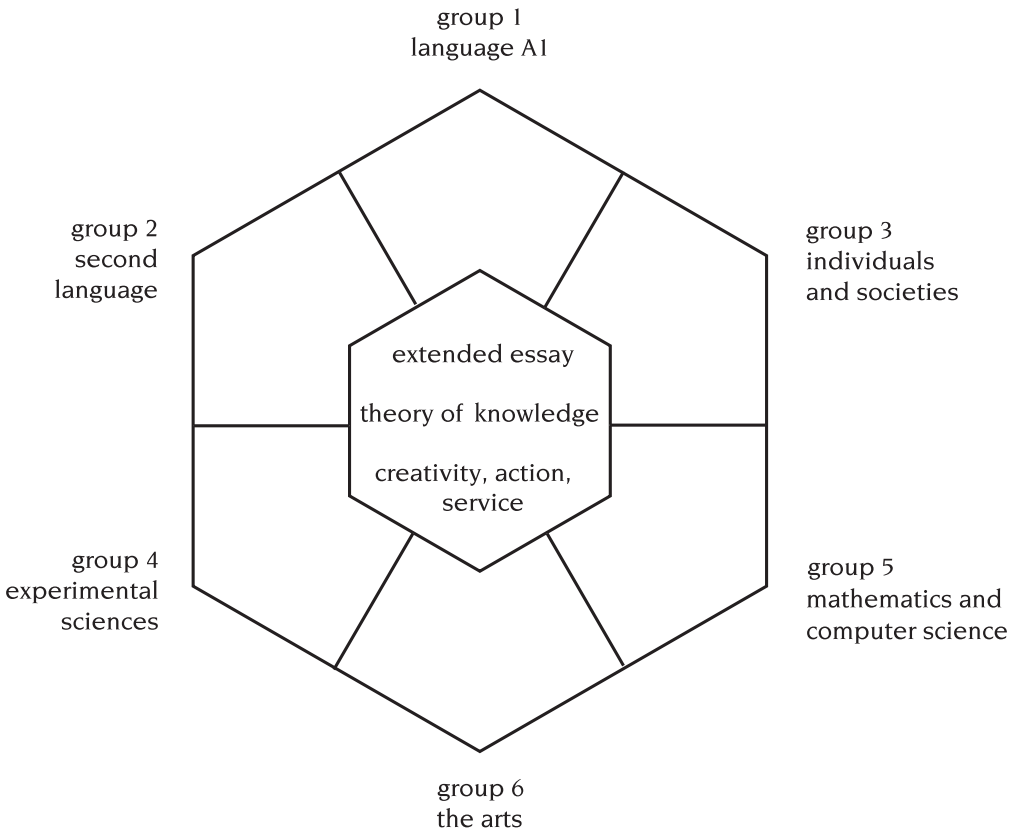
Printed in the United Kingdom by the International Baccalaureate Organization, Cardiff.

CONTENTS

INTRODUCTION

The International Baccalaureate Diploma Programme (DP) is a rigorous pre-university course of studies, leading to examinations, that meets the needs of highly motivated secondary school students between the ages of 16 and 19 years. Designed as a comprehensive two-year curriculum that allows its graduates to fulfill requirements of various national education systems, the DP model is based on the pattern of no single country but incorporates the best elements of many. The DP is available in English, French and Spanish.

The programme model is displayed in the shape of a hexagon with six academic areas surrounding the core. Subjects are studied concurrently and students are exposed to the two great traditions of learning: the humanities and the sciences.



DP students are required to select one subject from each of the six subject groups. At least three and not more than four are taken at higher level (HL), the others at standard level (SL). HL courses represent 240 teaching hours; SL courses cover 150 hours. By arranging work in this fashion, students are able to explore some subjects in depth and some more broadly over the two-year period; this is a deliberate compromise between the early specialization preferred in some national systems and the breadth found in others.

Distribution requirements ensure that the science-orientated student is challenged to learn a foreign language and that the natural linguist becomes familiar with science laboratory procedures. While overall balance is maintained, flexibility in choosing HL concentrations allows the student to pursue areas of personal interest and to meet special requirements for university entrance.

Successful DP students meet three requirements in addition to the six subjects. The interdisciplinary theory of knowledge (TOK) course is designed to develop a coherent approach to learning that transcends and unifies the academic areas and encourages appreciation of other cultural perspectives. The extended essay of some 4,000 words offers the opportunity to investigate a topic of special interest and acquaints students with the independent research and writing skills expected at university. Participation in the creativity, action, service (CAS) requirement encourages students to be involved in creative pursuits, physical activities and service projects in the local, national and international contexts.

First examinations 2006

NATURE OF THE SUBJECT

Problem solving

Computer science involves solving problems using computers. Therefore a full understanding of logical problem solving is required as well as a detailed knowledge of how computers operate. Successful computerized systems result from: a clear understanding of the problem to be solved; appropriate use of hardware based on a detailed knowledge of its capabilities and limitations; efficient use of algorithms and data structures; thorough and logical design; careful testing and integration of all these components. Students of Diploma Programme computer science will be guided by problem-solving strategies that will be continually reinforced in their coursework. Initial stages of the process will involve identifying and defining the problem(s) to be solved using a computerized system. The problem will be broken down (decomposed) into parts, with each part requiring a particular solution. From this problem definition, the student will construct appropriate algorithms to create a solution. The emphasis should be on the use of a logical approach and analytical thinking while using a computer to solve problems.

Java

Students are expected to acquire mastery of the specified aspects of Java. Suitable mechanisms include encapsulation, polymorphism and inheritance, although other structured approaches are possible. Mastery of a particular aspect (or mechanism) of computer science is defined as the ability to use that aspect appropriately for some non-trivial purpose that is well documented. Mastery will be demonstrated through work submitted in the program dossier.

The courses

The computer science standard level (SL) course focuses on software development, fundamentals of computer systems and the relationship between computing systems and society. The higher level (HL) course encompasses all these elements but is extended to include: computer mathematics and logic; advanced data structures and algorithms; further system fundamentals; and file organization.

RESOURCES

Facilities required

The facilities considered **essential** to teaching computer science are access to:

- one personal computer (workstation) per student while carrying out programming work, both during normal class time and out-of-class hours
- a compiler and editor for Java including debugging tools
- a printer
- the Internet.

Facilities recommended

The facilities **recommended**, but not considered essential, are access to:

- a network
- a variety of other equipment/devices (for example, scanner, CD-ROM).

Facilities not required

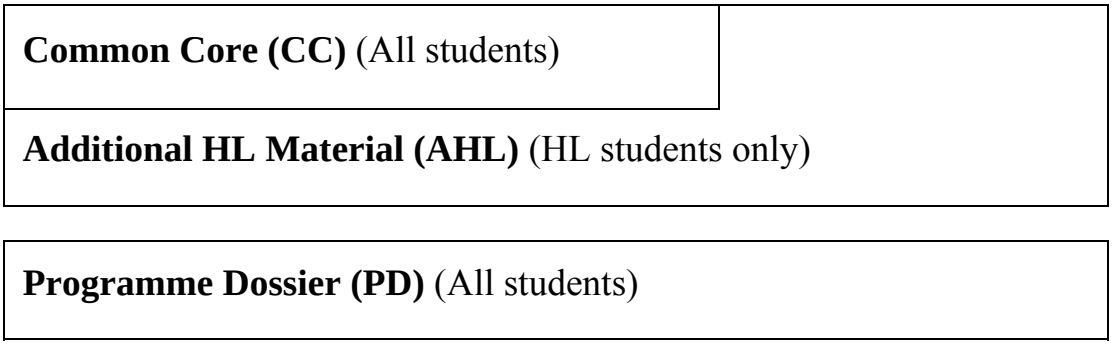
The facilities **not required** are access to:

- control/robotic devices
- CASE tools.

CURRICULUM MODEL

Both standard level (SL) and higher level (HL) students must study a **common core** (CC) of material and must demonstrate problem-solving skills and mastery of various aspects of computer science by completing a **program dossier** (PD). In addition, HL students must study **additional higher level material** (AHL) that fulfills two functions: it extends some topics in the CC, to give greater depth, and at the same time, introduces new topics to provide greater breadth.

The existence of a common core will allow teachers to teach SL and HL students together (where necessary), for at least part of the time. This curriculum model should **not** be taken to imply that it is intended that SL and HL students should be taught together. The IBO does **not** support the joint teaching of students at different levels, which does not provide the greatest educational benefit for either level, but recognizes that it can be a necessity in some schools.



Teaching time

The teaching time that should be allocated to this model is in accordance with the Diploma Programme requirement of 150 hours for SL courses and 240 hours for HL courses. The distribution is as follows:

Part of model	Target	Class time
Common core	all students	125 hours
Additional HL material	HL students only	80 hours
Program dossier	SL students	25 hours
	HL students	35 hours

The hours stated do **not** include time out of class for access to a computer (with the appropriate compiler and editor) required for the development of programs related to the syllabus and in connection with the program dossier.

AIMS

The aims of all courses in group 5 are to enable students to:

- appreciate the multicultural and historical perspectives of all group 5 subjects
- enjoy the courses and develop an appreciation of the elegance, power and usefulness of the subjects
- develop logical, critical and creative thinking
- develop an understanding of the principles and nature of the subject
- employ and refine their powers of abstraction and generalization
- develop patience and persistence in problem solving
- appreciate the consequences arising from technological developments
- transfer skills to alternative situations and to future developments
- communicate clearly and confidently in a variety of contexts.

OBJECTIVES

At the end of either the standard level (SL) or higher level (HL) computer science courses, students will be expected to fulfill the following objectives.

1. **Demonstrate an understanding of:** terminology, concepts, processes, structures, techniques, principles, systems and consequences (social significance and implications) of computing.
2. **Apply and use:** terminology, concepts, processes, structures, techniques, principles and systems of computing.
3. **Analyse, discuss and evaluate:** terminology, concepts, processes, structures, techniques, principles, systems and consequences (social significance and implications) of computing.
4. **Construct:** processes, structures, techniques and systems of computing.

OBJECTIVES AND ACTION VERBS

These apply to the assessment statements and computer science examination questions. Teachers are advised to ensure that their students are familiar with these definitions. Students may be guided on the meaning of the action verbs used in specific questions.

Objective 1

Define	Give the precise meaning of a word or phrase as concisely as possible
Draw	Represent by means of pencil lines. Add labels unless told not to do so. (Sometimes objective 2.)
State	Give a specific name or other brief answer. No supporting argument or calculation is necessary.

Objective 2

Apply	Use an idea, equation, principle, theory or law in a new situation. (Sometimes objective 3.)
Calculate	Find an accurate answer using mathematical or other formal methods. Show the working unless instructed not to do so. "Convert", "Express" or "Simplify" may be used to refer to specific forms of calculation. (Sometimes objective 3.)
Describe	Give a detailed account, including all the relevant information.
Estimate	Find an approximate answer, generally using mathematical methods.
Identify	Find an answer from a number of possibilities. (Sometimes objective 3.)
Outline	Give a brief account or summary, including essential information only.
Trace	Follow and record the action of an algorithm. (Sometimes objective 3.)

Objective 3

Analyse	Interpret information to reach conclusions.
Compare	Give an account of similarities and differences between two (or more) items, referring to both (all) of them throughout. Comparisons can be given using a table.
Discuss	Give an account including, where possible, a range of arguments and assessments of the relative importance of various factors or comparison of alternative hypotheses or ideas.
Explain	Give a clear account including causes, reasons or mechanisms.
Evaluate	Assess the implications and limitations. (Sometimes objective 4.)

Objective 4

Construct	Formulate and/or assemble information in a logical manner.
Design	Produce a plan, object, simulation or model.
Determine	Find the only possible answer. (Sometimes objective 2.)
Suggest	Propose a solution, hypothesis or other possible answer.

SYLLABUS OUTLINE

Computer science

Common core (HL and SL students) 125 hrs

Topic 1—Systems life cycle and software development 35 hrs

1.1	**The systems life cycle**	**8 hrs**
1.2	**Systems analysis**	**4 hrs**
1.3	**Systems design**	**4 hrs**
1.4	**Social significance and implications of computer systems**	**5 hrs**
1.5	**Software life cycle**	**2 hrs**
1.6	**Software design**	**8 hrs**
1.7	**Documentation**	**4 hrs**

Topic 2—Program construction in Java 50 hrs

Topic 3—Computing system fundamentals 37 hrs

3.1	**Language translators**	**2 hrs**
3.2	**Computer architecture**	**12 hrs**
3.3	**Computer systems**	**5 hrs**
3.4	**Networked computer systems**	**8 hrs**
3.5	**Data representation**	**6 hrs**
3.6	**Errors**	**2 hrs**
3.7	**Utility software**	**2 hrs**

Case study 3 hrs

Program dossier

Standard level (SL)	25 hrs
Higher level (HL)	35 hrs
Additional HL material (HL students only)	80 hrs
Topic 4—Computer mathematics and logic	11 hrs
4.1 Number systems and representations	**6 hrs**
4.2 Boolean logic	**5 hrs**
Topic 5—Abstract data structures and algorithms	41 hrs
5.1 Fundamentals	**3 hrs**
5.2 Static data structures	**8 hrs**
5.3 Dynamic data structures	**14 hrs**
5.4 Objects in problem solutions	**6 hrs**
5.5 Recursion	**6 hrs**
5.6 Algorithm evaluation	**4 hrs**
Topic 6—Further system fundamentals	15 hrs
6.1 Processor configuration	**2 hrs**
6.2 Magnetic disk storage	**1 hr**
6.3 Operating systems and utilities	**2 hrs**
6.4 Further network fundamentals	**4 hrs**
6.5 Computer/peripheral communication	**6 hrs**
Topic 7—File organization	10 hrs
Case study (Extended study)	3 hrs

SYLLABUS DETAILS

Format of the syllabus

Each part of the syllabus provides the following information.

- **Topics:** These are numbered 1–3 (for CC) and 4–7 (for AHL).
- **Subtopics:** These are numbered 1.1, 1.2 and so on. Each has an estimated teaching time.
- **Assessment statements:** These are numbered 1.1.1, 1.1.2 and so on.
- **Teaching notes:** These appear in a separate column.
- **Assessment objectives (Obj):** These are indicated by 1, 2, 3 or 4. (See the objectives.)

Assessment statements

The assessment statements form an examination syllabus, **not** a teaching syllabus, and are intended to prescribe to examiners what can be assessed by means of the written examinations. Each statement is classified according to the computer science assessment objectives **1**, **2**, **3** or **4** using appropriate action verbs. The objectives are relevant for balance within the syllabus and the examinations.

The **action verbs** are important because they give guidance to students and teachers about the depth and breadth required. It is important that students are aware of the meaning of these action verbs so that they know precisely the intent of examination questions and what will be expected 0 TD0.16(y)-7.3( )57, mea

Time allocation

The recommended teaching time for a Diploma Programme SL subject is **150** hours; the corresponding time for an HL subject is **240** hours. The time allocations given in the syllabus outline and the syllabus details are approximate, and are intended to suggest how time might be divided between the various topics and the program dossier. However, the exact time spent on each topic will depend on a number of factors, including students' background knowledge and their level of preparedness.

For computer science SL it is expected that **25** hours' teaching time will be spent on work for the program dossier; this value increases to **35** hours for HL. These hours do not include time out of class for access to a computer (with the appropriate compiler/interpreter) required for the development of programs related to the syllabus and in connection with the program dossier. (See the curriculum model.)

Use of calculators

Calculators can be used in the course but are not allowed in examinations.

Teacher support materials

A variety of teacher support materials to accompany this guide is being produced. These will include guidance for teachers marking program dossiers, and specimen examination papers and markschemes.

Common Core

Topic 1—Systems life cycle and software development

Students should understand the tasks that a systems analyst would perform when considering a situation that may be computerized. These and subsequent tasks included within the systems life cycle are covered in this topic. An understanding and mastery of some of these aspects is expected to be reflected in the program dossier.

Students should learn to analyse and solve problems, not just to write programs. The software life cycle involves several stages, and students are expected to be involved at some level in all stages. Good systems analysis should include investigation, data collection, careful planning and thorough documentation. If the problem is analysed properly, the implementation will be easier and more successful.

1.1—The systems life cycle 8 hrs

	Assessment statement	Teaching note	Obj
1.1.1	Outline the systems life cycle in terms of the stages: analysis, design, implementation, operation and maintenance.	Other models are acceptable as long as they emphasize the cyclical nature of the problem-solving process.	**2**
1.1.2	Explain the importance of collecting data during the analysis stage.		**3**
1.1.3	Compare methods of data collection.	Examples are: interviewing current users and domain experts, constructing questionnaires, observing current systems and studying prospective user-based documentation.	**3**
1.1.4	Describe the production of a requirements specification during the analysis stage.	This may include: a definition of inputs and outputs, a list of tools, facilities, people available for developing the solution, and a schedule for the next stages of the project.	**2**
1.1.5	Outline the features of a feasibility report.	A feasibility report may be produced during the analysis phase, the design phase, or both. It may include: a brief description of the proposed system; estimated costs; economic, technical and legal responsibility; and a possible completion date.	**2**

Topic 1—Systems life cycle and software development (continued)

	Assessment statement	Teaching note	Obj
1.1.6	Compare the advantages and disadvantages of alternative solutions in the design stage. This should include both hardware and software solutions.	Several possible solutions should be considered and evaluated by asking questions such as: What kind of output should be produced? Where will the input data come from and how will it be entered? Should the system be centralized or networked? Should computers be used at all? Should standard software packages be used? How much customization is desirable? The emphasis should be on modular organization. The human–computer interface should be considered.	**3**
1.1.7	Discuss methods of testing systems, the importance of proper testing and the implications of inadequate testing.	Students must be able to propose suitable test data, with reasons, during the design and implementation stages.	**3**
1.1.8	Outline methods of implementing new systems.	Methods include: running old systems in parallel with new systems, direct changeover and phased introduction. Training implications and possible disruption during installation should be considered.	**2**
1.1.9	Outline the features and importance of maintaining systems.	Features to be considered include periodic reviews, performance evaluation and clear documentation, to facilitate modifications.	**2**

Topic 1—Systems life cycle and software development (continued)

1.2—Systems analysis 4 hrs

Students should learn to investigate and analyse problems at the system level before beginning to think about a solution (algorithms). Students should be able to read and construct system flowcharts.

	Assessment statement	Teaching note	Obj
1.2.1	Explain the importance of formulating a problem precisely.		**3**
1.2.2	Discuss the aspects that must be considered in a specified problem.	Students should realize that activities such as interviews, questionnaires and literature searches are required to discover the relevant aspects.	**3**
1.2.3	Identify the outcomes that an appropriate solution must produce to solve a specified problem.		**2**
1.2.4			

Topic 1—Systems life cycle and software development (continued)

	Assessment statement	Teaching note	Obj
1.3.3	Outline suitable means of data capture and output presentation for a system.		**2**
1.3.4	Design appropriate data structures to store data within a system.		**4**
1.3.5	State the hardware components that are appropriate for a system.		**1**
1.3.6	Outline a suitable interface between a system and users.		**2**
1.3.7	Analyse a systems flowchart that represents a complete system.		**3**
1.3.8	Construct a systems flowchart to represent a complete system.	Symbols that students should use appear in appendix 3.	**4**

1.4—Social significance and implications of computer systems 5 hrs

	Assessment statement	Teaching note	Obj
1.4.1	Discuss the social and economic implications of the installation of new systems.	See 1.1.8 for installation methods to be considered.	**3**
1.4.2	Discuss the social significance and implications of the widespread use of computers in society.	The social significance must be treated by reference to economic, political, cultural and environmental consequences. These include: effects on employment (resulting in changes to the working environment, retraining and so on); computers (hacking, viruses and so on); ethical and legal requirements; data storage (preserving privacy, data protection and so on); software users (copyright, software licensing and so on).	**3**
1.4.3	Discuss current trends in computer systems and the consequences of these trends.		**3**

Topic 1—Systems life cycle and software development (continued)

1.5—Software life cycle 2 hrs

	Assessment statement	Teaching note	Obj
1.5.1	Outline the major stages in the software life cycle.	One model includes: systems analysis, leading to a precise statement of the problem that needs solving (a requirements specification); software design; program construction, including testing and debugging; installation and operation; and maintenance. Other models are acceptable, as long as they emphasize the cyclical nature of the life cycle.	**2**
1.5.2	Explain why software production is normally cyclical.	Students should understand that computer systems are used over long periods of time. The software in these systems requires periodic improvement. After the original design and implementation, further analysis, redesign and restructuring are required to accommodate changing needs. This will continue through many cycles of analysis, design, implementation and use.	**3**

1.6—Software design 8 hrs

	Assessment statement	Teaching note	Obj
1.6.1	Outline the data required to solve a problem that the students have not encountered before, including data file formats, input and output requirements with appropriate user interfaces. For example, screens, OMR forms, report formats.		**2**
1.6.2	Discuss the advantages of modularity in designing a solution to problems.		**3**
1.6.3	Define the term prototyping.		**1**

Topic 1—Systems life cycle and software development (continued)

	Assessment statement	Teaching note	Obj
1.6.4	Outline the prototyping approach to systems design and development.	Prototyping can be done at many levels of sophistication. For the purposes of this course prototyping is limited to the presentation of a preliminary solution that may not be functional.	**2**
1.6.5	Discuss the advantages to end-users and systems designers of using the prototyping approach.	Prototyping can be used with end-users for the purpose of obtaining feedback at an early stage in the design process. Prototyping can be used by systems designers to investigate alternative solutions to a problem.	**3**
1.6.6	Outline the efficiency of a solution in terms of storage requirements, memory requirements, and speed.	Only a qualitative treatment or a specific calculation is expected; "O" or BigO notation is required only at HL. (See 5.6, Algorithm evaluation.)	**2**
1.6.7	Outline how programs can be tested and debugged.	Testing implies tracing sections of an algorithm, including responses to errors ("dry runs"), as well as the design of test cases, which are then executed. Students must be able to propose suitable test data, and give reasons. Debugging has the components of detecting, diagnosing and correcting errors shown up by testing.	**2**
1.6.8	Describe the role of tools in constructing, testing and debugging programs.	Ideally, students should use an integrated development environment (IDE) combining an editor, interpreter or compiler and debugging tools, but this is not a requirement.	**2**

Topic 1—Systems life cycle and software development (continued)

1.7—Documentation 4 hrs

	Assessment statement	Teaching note	Obj
1.7.1	Outline why documentation is needed at each stage of the systems life cycle.		**2**
1.7.2	Explain the features of documentation for design, programming and maintenance, that is, system documentation.	Students are required to document their problem-solving process according to the standards described in the guidelines for program dossiers. Program listings must also be thoroughly documented.	**3**
1.7.3	Explain the features of documentation for the user, that is, user documentation.	Students are required to write end-user instructions according to the standards described in the guidelines for the program dossier. Students should know that other user manuals may be needed (for example, on-line help systems and installation manuals where systems are installed by personnel other than end-users), but they are not required to write such documentation.	**3**

Topic 2—Program construction in Java

2.1—Program construction in Java 50 hrs

Discussion of the material in this subtopic will play a major role in the development of program dossiers. While 50 hours have been allocated it should be noted that some of the 25 hours allocated as teacher contact time will also be used in discussion of these aspects. The high-level language must be Java syntax as specified in appendix 2.

	Assessment statement	Teaching note	Obj
2.1.1	Apply the following high-level language constructs appropriately in order to implement a software design expressed in Java. • Declare variables and types with appropriate scope, distinguishing between private and public identifiers. • Define and apply user-defined objects. • Format output in a user-friendly manner. • Construct and calculate arithmetic, relational and Boolean expressions (only `and`, `or`, `not`) using appropriate operators (&&, || and !) and taking into account their precedence. • Construct and calculate the value of modulo arithmetical expressions “mod”, “div” using appropriate operators (%, /) and taking into account their precedence. • Implement the remaining algorithm constructs in Java: arrays, objects, selection constructs (branching), file operations, iteration constructs (looping), sentinels and flags. • Use built-in subprograms, including those of the Java foundation classes specified in appendix 2. • Define and apply user-defined methods. • Demonstrate an understanding of method signatures. • Demonstrate an understanding of the use of parameters, including object and primitive parameter passing and return values. • Demonstrate an understanding of the scope of Java identities, restricted to the keywords private and public. • Define primitive, class, object, data member, method, method signature and constructor.		**3**

Topic 2—Program construction in Java (continued)

	Assessment statement	Teaching note	Obj
2.1.2	Apply appropriate data types and data structures to solve a problem that students have not encountered before.	Required data types are integer, real, character and Boolean. Required data structures are strings, one-dimensional arrays, two-dimensional arrays, records and files.	**3**
2.1.3	Describe the nature and function of the data types and data structures stated in 2.1.2.		**2**
2.1.4	Trace algorithms in Java.	See appendix 2. Examination questions will always use Java whenever code needs to be displayed; therefore students must be able to understand algorithms presented in this language. The algorithms may be either the standard algorithms in the syllabus, or algorithms of equivalent complexity that the students have not seen before. The algorithms may use any of the data types and structures listed in 2.1.2.	**2**
2.1.5	Evaluate algorithms written in Java with respect to efficiency, correctness and appropriateness for a task.	See note in 2.1.4.	**3**
2.1.6	Construct algorithms in Java.	See note in 2.1.4.	**4**
2.1.7	Explain the need for searching and sorting.		**3**
2.1.8	Apply sequential (linear) and binary search algorithms, selection and bubble sort algorithms to problems, including some not encountered before.	Searching and sorting provide good examples for studying the design, development and analysis of algorithms. Students should be able to discuss the appropriate circumstances for the use of each algorithm. In examinations they may be given descriptions of other algorithms to be developed.	**3**
2.1.9	Compare the efficiency of the specific searching and sorting algorithms mentioned in 2.1.8.		**3**
2.1.10	Discuss the efficiency of specific searching and sorting algorithms.	BigO notation is not required at SL.	**3**

Topic 2—Program construction in Java (continued)

	Assessment statement	Teaching note	Obj
2.1.11	Describe syntax errors, logic errors, and run-time errors.	Overflow, underflow and truncation errors may arise during program development and so they may be discussed, but they will not be examined at SL.	**2**

Topic 3—Computing system fundamentals

This topic covers computer systems (their hardware and software) and how they interact.

3.1—Language translators 2 hrs

	Assessment statement	Teaching note	Obj
3.1.1	Define syntax and semantics.		**1**
3.1.2	Describe the function of high-level language translators.	The translators should be limited to interpreters and compilers.	**2**
3.1.3	Outline the use of software development tools.	Examples include: database management systems, macros, CASE tools and simple language translators (interpreters and compilers are not suitable examples in this context), HTML editor, web page editor, code editor, visual IDE.	**2**

3.2—Computer architecture 12 hrs

	Assessment statement	Teaching note	Obj
3.2.1	Outline the structure of the central processing unit (CPU) including the functions of the control unit (CU), the arithmetic and logic unit (ALU), primary memory and address buses.	Students are expected to be able to reproduce a basic diagram illustrating the CPU and to know that each location in primary memory has a unique address.	**2**
3.2.2	Outline the meaning of the terms bit (b) and byte (B) and their derivatives.	Students must understand that everything in a computer is held and processed in binary, hence the relation between bits, bytes and so on in powers of 2. For example 1kilobyte = 2^{10}. They should be familiar with the prefixes T, G, M, k and their use in computer science measure. They must be able to apply the prefixes T, G, M and k to bits and bytes. For example TB (terabytes), Gb (gigabits), MB (megabytes).	**2**
3.2.3	Outline the meaning of the terms word, register and address and their use in the storage of data and instructions.	The study of specific registers is not required.	**2**

Topic 3—Computing system fundamentals (continued)

	Assessment statement	Teaching note	Obj
3.2.4	Outline the steps in the machine instruction cycle: fetch, decode, execute and store.	A single processor model is sufficient. The study of a specific CPU is not required.	**2**
3.2.5	Outline the characteristics of primary memory and the difference between volatile and non-volatile memory.	Students must understand the function of RAM, ROM and cache memory and their typical sizes (in bytes). The way in which virtual memory can be used to expand primary memory must be understood but details of paging are not needed.	**2**
3.2.6	Outline the characteristics of secondary memory and define sequential and direct access.	Secondary memory should refer to flash memory, disks, CDs and DVDs and tape. Students must know the type of access of the above secondary memory media. They should also be able to give an application of each type and justify its use for this application.	**2**
3.2.7	Outline the function of a microprocessor designed to perform one or a limited number of functions (within a car, washing machine and so on).	The need for different types of memory in a microprocessor must be understood. Students must be able to quote at least one example of the use of a microprocessor and state the inputs and outputs.	**2**
3.2.8	Discuss the features, advantages, disadvantages and applications of specific input and output devices and the media used by each.	Students must know the following features: mouse, keyboard, touch screen, optical character recognition (OCR), magnetic ink recognition (MICR), scanners (page, mark sense and barcode), LCD panels, speech recognition, sensors, digital cameras, graphics tablets, printers, plotters, monitors, robotics, sound. Technical details are not required unless introduced in the case study.	**3**
3.2.9	Outline recent developments in computer system architecture including processor architecture, primary memory technologies and secondary memory devices.	Technical details are not required unless introduced in the case study.	**2**

Topic 3—Computing system fundamentals (continued)

3.3—Computer systems 5 hrs

	Assessment statement	Teaching note	Obj
3.3.1	Define the term "operating system".	Knowledge of specific operating systems is not required.	**1**
3.3.2	Outline the functions of operating systems.	Functions include: communicating with peripherals; coordinating concurrent processing of jobs; memory management, resource monitoring, accounting and security; program and data management; providing appropriate user interfaces.	**2**
3.3.3	Discuss the characteristics of various computer systems including single users and multi-users, in both single-tasking and multi-tasking environments.	The terms multi-access and multi programming should be understood but details of the way in which they are managed will not be examined.	**3**
3.3.4	Compare the characteristics and applications of different kinds of computers.	Personal computers, portable computers, mainframes and supercomputers should be considered. Characteristics must include: primary and secondary memory size; input/ouput (I/O) devices; environment (size, convenience, where it is used); cost, users (multi- or single-); and processor (word length, bus size and frequency).	**3**
3.3.5	Outline the principal characteristics of batch processing, online (interactive) processing and real-time processing.		**2**
3.3.6	Outline applications that use each of the processing methods in 3.3.5: batch processing (payroll and bank cheque processing); interactive (online) processing; word processing; computer games; real-time processing (air traffic control and monitoring of patients in hospital intensive care).		**2**
3.3.7	Explain the relationship between master and transaction files.	This should relate to the examples in 3.3.6.	**3**
3.3.8	Discuss the reliability of the system including the implications of failure.	The need for, and use of, backing-up strategies, mirrored systems and the utilities in 3.7.	**3**

Topic 3—Computing system fundamentals (continued)

3.4—Networked computer systems 8 hrs

	Assessment statement	Teaching note	Obj
3.4.1	Define local area network (LAN), wide area network (WAN), server and client.		**1**
3.4.2	Explain basic network topologies.	Students must be able to explain and illustrate star and bus networks as well as hybrids involving both these networks.	**3**
3.4.3	Explain the hardware required in networking.	Hardware should include communications links (cables, microwave, fibre optics and so on) hub, switch, node and router.	**3**
3.4.4	Define the terms "standard protocol", "data integrity" and "data security" in the context of data transmission across a network.	Students must know that standard protocols are a set of rules that are internationally recognized in the transmission of data. The difference between data security and data integrity must also be recognized. Students do not need to know specific or technical details such as the ISO (OSI) system of layers, TCP/IP and so on.	**1**
3.4.5	Explain the software involved in networking.	Students must understand the role of communications software in connecting local and wide area networks and the need to deal with protocols and data security.	**3**
3.4.6	Describe suitable methods to ensure data integrity in the transmission of data.	Error-checking codes such as check sums (block character checks) and parity checks must be understood. The reasons for re-transmission should be understood. The quality of communication lines should be considered.	**2**
3.4.7	Describe suitable methods to ensure data security.	Students should understand the concept of data encryption but do not need to give algorithmic details. They must understand the need for, and use of, passwords, physical security and different levels of access (permissions) for different users.	**2**

Topic 3—Computing system fundamentals (continued)

	Assessment statement	Teaching note	Obj
3.4.8	Discuss the need for speed in data transmission, and how speed can be enhanced.	Students must know that documents and graphics files can be sent in different formats and the format affects the speed of transmission. Common formats such as JPEG and BMP should be known. The principles of data compression should be considered but details of methods are not required.	**3**
3.4.9	Discuss networking applications and the implications of networking for organizations, including internal communications, electronic mail, e-commerce, conferencing and distributed processing.	The use of LANs, public and private WANs and the Internet should be considered.	**3**
3.1.10	Outline the functions of a web browser and search engine including displaying an HTML page, following hyperlinks and searching on key words.	Specific names of browsers and search engines are not required.	**2**

3.5—Data representation 6 hrs

	Assessment statement	Teaching note	Obj
3.5.1	Outline the use of binary to represent data.	Students must understand the relationship between number of digits and number of patterns available (2^n, for example: 4-bit colour representation allows 16 colours; a 32-bit address bus can address 4GB RAM). The different features of ASCII and Unicode should be known but students are not expected to know the specific representations of characters.	**2**
3.5.2	Outline the need for standard formats for storing documents and files.	Link with 3.4.8 and 3.4.9.	**2**
3.5.3	Express numbers in the bases: decimal, binary and hexadecimal.		**2**
3.5.4	Convert integers between the bases specified in 3.5.3 (maximum 8 bits).		**2**

Topic 3—Computing system fundamentals (continued)

	Assessment statement	Teaching note	Obj
3.5.5	Apply binary notation to represent integers, both positive and negative, using the method-of-two's complement.		**2**
3.5.6	Define analogue data and digital data.		**1**
3.5.7	Outline the need for the interconversion of data between analogue and digital formats for computer processing.	Students need to understand the need for conversion between data for processing, for example, sensors and modems.	**2**
3.5.8	Discuss two applications that require conversion of data between analogue and digital formats including temperature sensing.	Teachers are free to choose the second application. Other software examples include: speech recognition, light detection, image processing, OCR software.	**3**

3.6—Errors 2 hrs

	Assessment statement	Teaching note	Obj
3.6.1	Describe the following causes of errors with reference to an application in each case: data entry, accidental, deliberate, software and hardware.		**2**
3.6.2	Outline methods of detection and prevention for each of the errors in 3.6.1.	Verification and validation should be understood. Check digits and hash totals should be explained. The use of modulo operators (mod, div) in constructing check digits should be understood.	**2**
3.6.3	Describe methods of recovery from an error.	Re-input, re-transmission and restoring from backups should be considered. Error-correcting algorithms are not required.	**2**

Topic 3—Computing system fundamentals (continued)

3.7—Utility software 2 hrs

	Assessment statement	Teaching note	Obj
3.7.1	Outline the main function(s) of the following software utilities: data compressors, virus software, file managers, defragmentation software.	The required file manager functions are: copy, delete, format, find, create folder/directory, archive, print, back-up, rename and restore. The fact that files are not stored contiguously only needs to be dealt with in outline, in order to understand why defragmentation software is required. Technical details are not required.	**2**
3.7.2	Discuss the need for each of the utilities in 3.7.1.		**3**

Additional HL material

Topic 4—Computer mathematics and logic

Computer science is not a mathematics course. However, the following topics allow students to understand the basic principles of computer architecture, to understand the fundamental causes of many common errors, to design simple circuits, and to construct some common algorithms requiring mathematical techniques.

4.1—Number systems and representations 6 hrs

	Assessment statement	Teaching note	Obj
4.1.1	Calculate in the bases specified in 3.5.3.	For binary and hexadecimal calculations, only addition is required.	**3**
4.1.2	State the mantissa and exponent of a binary number in floating-point representation. Relate this to scientific notation in decimal.	For negative binary numbers in integer and real formats, only the method-of-two's complement is required.	**1**
4.1.3	Apply binary notation to represent real numbers.	Both fixed-point and floating-point representations are required. Students should be able to calculate the range of normalized floating-point numbers given a specific representation. Issues such as the need for normalization and the loss of precision should be understood.	**2**
4.1.4	Discuss the advantages and disadvantages of integer and floating-point representations.		**3**
4.1.5	Define truncation error, underflow error and overflow error.		**1**
4.1.6	Outline three situations, with each one providing an example of when and where one of the errors in 4.1.5 can occur. Each situation should show a different error, that is all three errors should be described.		**2**

Topic 4—Computer mathematics and logic

4.2—Boolean logic 5hrs

<table>
<tr><th></th><th>Assessment statement</th><th>Teaching note</th><th>Obj</th></tr>
<tr><td>4.2.1</td><td>Define the Boolean operators and, or, not, nand, nor and xor, by drawing the appropriate truth table.</td><td></td><td>1</td></tr>
<tr><td>4.2.2</td><td>Construct Boolean expressions using the operators in 4.2.1.</td><td><table><tr><th>Operator</th><th>Symbol</th></tr><tr><td>and</td><td>•</td></tr><tr><td>or</td><td>+</td></tr><tr><td>not</td><td>(overbar)</td></tr><tr><td>xor</td><td>⊕</td></tr></table>
For example, $\left(A \oplus \overline{B}\right) \cdot \left(\overline{C + D}\right)$.

This can be written in words as: (A xor not B) and (C nor D).</td><td>4</td></tr>
<tr><td>4.2.3</td><td>Calculate the values of a Boolean expression using truth tables.</td><td>A maximum of three inputs will be expected. Include the use of truth tables to determine whether two Boolean expressions are logically equivalent.</td><td>3</td></tr>
<tr><td>4.2.4</td><td>Convert Boolean expressions into simpler forms.</td><td>A maximum of three inputs will be expected. Conversions may be done “algebraically” (using identities such as $x + 1 = 1$ and De Morgan’s laws) or by using Karnaugh maps, Venn diagrams or any other appropriate method.</td><td>2</td></tr>
<tr><td>4.2.5</td><td>Construct a simple logic circuit that corresponds to a given Boolean expression by using standard logic gates.</td><td></td><td>4</td></tr>
</table>

Topic 4—Computer mathematics and logic (continued)

	Assessment statement	Teaching note	Obj
4.2.6	Construct a Boolean expression that corresponds to a given logic circuit.		**4**
4.2.7	Explain the function of a given circuit.		**3**

Topic 5—Abstract data structures and algorithms

The Java programming language provides some standard data structures (such as arrays or files) that are adequate for many standard problems. Other problems require further data types to represent more complex structures, improve algorithm efficiency, or provide for more sophisticated memory management.

Although Java implements many different types of container class for the convenience of programmers, students are expected to be able to develop their own ADTs from first principles.

Higher level students must demonstrate mastery of some of these techniques in the program dossier and should be able to use any of these techniques during the examination. This topic extends several aspects of topics 1 and 2.

5.1—Fundamentals 3 hrs

	Assessment statement	Teaching note	Obj
5.1.1	Define operator (unary and binary), identifier, operand, actual parameter (argument), formal parameter, infix notation, postfix notation and prefix notation.		**1**
5.1.2	Define stack, queue and binary tree.		**1**
5.1.3	Discuss the features and appropriate usage of stacks including: parameter storage, interrupt handling, evaluation of arithmetic expressions and storage of subprogram return addresses.		**3**
5.1.4	Discuss the features and appropriate usage of queues including: keyboard queues, print queues and customer queue simulations.		**3**
5.1.5	Discuss the features and appropriate usage of binary trees including: storing search keys, decision trees and file systems.		**3**

Topic 5—Abstract data structures and algorithms (continued)

5.2—Static data structures 8 hrs

Arrays are covered at length in the common core. This subtopic should be considered as an extension.

	Assessment statement	Teaching note	Obj
5.2.1	Trace algorithms that perform a quicksort on linear arrays.		**2**
5.2.2	Construct algorithms that perform a quicksort on linear arrays.		**4**
5.2.3	Construct a hash table including the generation of addresses using modulo arithmetic and the handling of clashes by locating next free space.	Students will be given a hash algorithm and a set of keys or records that will be allocated memory locations by employing the algorithm.	**4**
5.2.4	Trace algorithms that implement a stack in an array.		**2**
5.2.5	Construct algorithms that implement a stack in an array.	This includes: to initialize a stack, to test for an empty or a full stack, to push a data item, to pop a data item and to display the top data item. All operations must protect against overflow and underflow.	**4**
5.2.6	Trace algorithms that implement a queue in an array.		**2**
5.2.7	Construct algorithms that implement a queue in an array.	This includes: to initialize a queue, to test for an empty or a full queue, to add a data item to the rear of a queue (enqueue), to remove a data item from the front of a queue (dequeue) and to display the data item at the front of a queue. Algorithms must include linear and circular implementation. All operations must protect against overflow and underflow.	**4**

Topic 5—Abstract data structures and algorithms (continued)

5.3—Dynamic data structures 14 hrs

	Assessment statement	Teaching note	Obj
5.3.1	Define object reference.		**1**
5.3.2	Construct algorithms that use reference mechanisms.		**4**
5.3.3	Discuss the features and appropriate usage of single, double and circular linked lists.		**3**
5.3.4	Outline and illustrate how links operate logically.		**2**
5.3.5	Trace algorithms to implement linked lists.		**2**
5.3.6	Construct algorithms to implement linked lists.	This includes: initialize, add objects, delete objects, find tail object, perform linear search and insert objects into a list. All operations must protect against null pointer exceptions.	**4**
5.3.7	Trace algorithms that implement a dynamic stack using references.		**2**
5.3.8	Construct algorithms that implement a dynamic stack using references.	Students must recognize the difference between this and the static representation of stacks. See notes in 5.2.5.	**4**
5.3.9	Trace algorithms that implement a dynamic queue using references.		**2**
5.3.10	Construct algorithms that implement a dynamic queue using references.	Students must recognize the difference between this and the static representation of a queue. See also notes in 5.2.7.	**4**
5.3.11	Define parent, left-child, right-child and subtree.		**1**
5.3.12	Trace algorithms to implement binary trees.		**2**

Topic 5—Abstract data structures and algorithms (continued)

	Assessment statement	Teaching note	Obj
5.3.13	Construct algorithms to implement binary trees.	This includes: initialize, add objects, traverse (pre-order, in-order and post-order). All tree traversals must be implemented recursively. See 5.5.	**4**
5.3.14	Outline and illustrate the logical representation of dynamic data structures.		**2**

5.4—Objects in problem solutions 6 hrs

The scope of the topic is limited to features exemplified in Java. (See appendix 2.)

	Assessment statement	Teaching note	Obj
5.4.1	Outline the features of an object.	This should be limited to the following definition. An object is a combination of data and the operations that can be performed in association with that data. Each data part of an object is referred to as a data member while the operations can be referred to as methods. The current state of an object is stored in its data members and that state should only be changed or accessed through the methods. Common categories of operations include: the construction of objects; operations that either set (mutator methods) or return (accessor methods) the data members; operations unique to the data type; and operations used internally by the object.	**2**
5.4.2	Explain the basic features and advantages of encapsulation.	Encapsulation is the combination of data and the operations that act on the data into a single "program unit" called an object. The advantages are that it allows for information and data hiding.	**3**
5.4.3	Explain the basic features and advantages of information and data hiding.	Once encapsulated into an object both the data members and the details of the implementation of the member functions can be hidden. This allows the object to be used at an abstract level.	**3**

Topic 5—Abstract data structures and algorithms (continued)

	Assessment statement	Teaching note	Obj
5.4.4	Explain the basic features and advantages of polymorphism.	Polymorphism describes the situation in which the same operation can be applied to different objects, with each object behaving appropriately. The concepts of templates, virtual member functions and operator overloading are not required. Polymorphism allows objects to be used intuitively and it simplifies coding by making it generic.	**3**
5.4.5	Explain the basic features and advantages of inheritance.	Inheritance allows one object to be derived from another. The derived object has all the data members and member functions of the original object and any additional data member or member functions that are defined within it. Even previously defined functionality may be redefined with the appropriate functionality applied to the particular object that invokes it. In Java, all classes are subclasses of the object class. When functions (including constructors) are redefined in a derived object, they completely override the original function. Inheritance in Java is limited to one object being derived from another (one level of inheritance). Multiple inheritance is not supported by the Java language.	**3**
5.4.6	Trace an algorithm that includes objects.	This will include recording the behaviour and state of the objects.	**2**

5.5—Recursion 6 hrs

	Assessment statement	Teaching note	Obj
5.5.1	Define recursion.		**1**
5.5.2	Discuss the advantages and disadvantages of recursion.	Students must understand that for some applications a recursive procedure is short and elegant, and that a recursive solution is ideally suited for some algorithms. However, recursion is not suitable for most algorithms as non-recursive ones are more efficient.	**3**

Topic 5—Abstract data structures and algorithms (continued)

	Assessment statement	Teaching note	Obj
5.5.3	Trace recursive algorithms.	All steps and calls must be shown clearly. Students may need to draw a tree.	**2**
5.5.4	Construct recursive algorithms.	This is limited to an algorithm that returns no more than one result and contains either one or two recursive calls to itself.	**4**
5.5.5	Implement the following constructs: self-referential classes and recursion.		**3**

5.6—Algorithm evaluation 4 hrs

	Assessment statement	Teaching note	Obj
5.6.1	State the efficiency of the following algorithms in BigO notation: a linear search is O(n), a bubble sort is $O(n^2)$, a quicksort is O(n log n), a binary search is O(log n) and a selection sort is $O(n^2)$, given a randomly distributed data set.	BigO notation is used to classify algorithm performance (speed). A sequential search is O(n), meaning that the time to search an array is proportional to the size of the array. However, a bubble sort requires nested loops and is therefore $O(n^2)$, so its time requirements are proportional to the square of the size of the list. Students should be aware that the efficiency of a given algorithm may depend on the distribution of the data, for example, a quicksort may deteriorate to $O(n^2)$ in the worst case.	**1**
5.6.2	Analyse the efficiency of algorithms (those in 5.6.1 and those of similar complexity), in terms of BigO notation and in terms of the storage requirements.	When students are presented with an algorithm that they have not encountered they must be able to write the BigO notation for the efficiency of that algorithm.	**3**

Topic 5—Abstract data structures and algorithms (continued)

	Assessment statement	Teaching note	Obj
5.6.3	Outline how data structures in this syllabus can be organized to suit the requirements of applications.	Students should consider the needs of different applications for data types and data structures. For example, **stacks** might be used to track changes to a word processing document whereas a **queue** might store data items being entered at the keyboard for subsequent processing in the order in which they arrived. Ordered **binary trees** and **hash tables** are frequently used to store key fields used for quick retrieval of items from an unordered **data file**.	**2**
5.6.4	Evaluate algorithms that use any of the data structures in this syllabus.	The algorithms may be standard algorithms mentioned in the syllabus, or algorithms of equivalent complexity that the students have not seen before.	**3**

Topic 6—Further system fundamentals

Actual computer system performance is affected by all the components of the system. Students need to know the functions of the individual components and the methods used in their interactions. This topic extends topic 3.

6.1—Processor configuration 2 hrs

	Assessment statement	Teaching note	Obj
6.1.1	Describe the functions of the following processor components: accumulator, instruction register and program counter.	Further details (or registers) are not required.	**2**
6.1.2	Explain the role of the above components in the execution of single instructions in the machine instruction cycle.		**3**
6.1.3	Describe the function of an interrupt register.		**2**
6.1.4	Describe how buses link the processor, the random access memory, the read-only memory and cache.		**2**

6.2—Magnetic disk storage 1 hr

	Assessment statement	Teaching note	Obj
6.2.1	Outline storage details with reference to blocking, sectors, cylinders and heads.		**2**
6.2.2	Describe access time in terms of latency (rotational delay), seek time and transfer time.		**2**

Topic 6—Further system fundamentals (continued)

6.3—Operating systems and utilities 2 hrs

	Assessment statement	Teaching note	Obj
6.3.1	Define operating system.	Knowledge of any specific operating system is not required.	**1**
6.3.2	Explain the functions of operating systems.	Students need to be aware that an operating system is a collection of programs that cover the following tasks: input/output (I/O) control, file maintenance, software/hardware interface, memory management, user interface, software execution control, security. Virtual memory must be included but knowledge of thrashing and paging is not required. This extends 3.3.2.	**3**
6.3.3	Outline the functions of linker, loader and library manager.		**2**

6.4—Further network fundamentals 4 hrs

	Assessment statement	Teaching note	Obj
6.4.1	Outline the role of the computers used in the separate type of networks: WAN, LAN and the Internet.	The roles of providers, servers and clients should be understood for each of these networks. Students should be able to select the appropriate type of network for a given situation. They must understand the role of gateways.	**2**
6.4.2	Describe the features of communications needed for networking.	Ethernet, public and private telephone lines, ISDN, ADSL, fibre optic and wireless methods should all be familiar and students should be able to select the most suitable method of communication in a given situation, and to state the advantages of each method. Technical details will not be required.	**2**
6.4.3	Describe packet switching.	Students need to be aware that when a message is dissembled into packets, the packets may take different paths and pass through different nodes to arrive at the same destination, and that packets can be discarded. Virtual circuits are not required.	**2**

Topic 6—Further system fundamentals (continued)

	Assessment statement	Teaching note	Obj
6.4.4	Outline the need for protocols in packet switching.	Students do not need to know technical details of TCP, IP, OSI, but must understand that protocols include essential information that allows packets to be reassembled at their destination according to the requirements of the receiving computer.	**2**
6.4.5	Explain the need for network security and describe how this can be achieved.	Emphasize the importance of protection within a LAN by giving layered access (for example, via permissions on certain areas) to different users, and marking files as read only. The need for a firewall to prevent intrusion from outside should be clear.	**3**

6.5—Computer/peripheral communication 6 hrs

	Assessment statement	Teaching note	Obj
6.5.1	Define port and handshaking.		**1**
6.5.2	Define direct memory access (DMA) and buffer.		**1**
6.5.3	Define interrupt and polling.		**1**
6.5.4	Explain how peripheral devices are controlled with reference to the printer, modem and disk drive.	This must include the use of buffers (including double buffering), interrupts and interrupt priorities, polling, direct memory access (DMA) and handshaking in these devices.	**3**
6.5.5	Compare the features of DMA, interrupt systems and polling systems.	Students must cover an event or external device interrupt as well as a polling system. Knowledge of specific interrupt codes is not required.	**3**
6.5.6	Compare serial transmission with parallel transmission.		**3**

Topic 7—File organization

A variety of file structures are commonly used in computer systems. Students must be familiar with several of the most common structures. This topic extends topic 1.

Current literature often appears confusing as the terminology relating to file structure and methods of accessing files is used inconsistently. The following table clarifies the specific terminology that is used in this syllabus, and that will be used in examinations.

File structure name	Structure details	Access method (searching)
Sequential file	Ordered or unordered records	Sequential access.
Partially-indexed file	Ordered records	Sequential access to index, followed by direct access to the first record in the group, then sequential access to find the desired record.
Fully-indexed file	Unordered records	Sequential access to the index, followed by direct access to the data file.
Direct access file	Unordered or ordered records	A calculation provides the address (location) of a record, followed by direct access to the record.

7.1—File organization 10 hrs

	Assessment statement	Teaching note	Obj
7.1.1	Define the term "key field".		**1**
7.1.2	Outline sequential file organization on unordered records and how records can be retrieved by using sequential access via the key field.		**2**
7.1.3	Outline sequential file organization on ordered records and how records can be retrieved by using sequential access via the key field.		**2**
7.1.4	Outline partially-indexed sequential file organization.	A partially-indexed sequential file has ordered records, with a separate but partial index. Students should be able to describe how records can be retrieved via access to the index, followed by direct access to the first record in a group, followed by sequential access to locate the desired record.	**2**

Topic 7—File organization

	Assessment statement	Teaching note	Obj
7.1.5	Outline fully-indexed file organization.	A fully-indexed file has unordered records, with a separate and complete index. Students should be able to describe how records can be retrieved via access to the index, followed by direct access in the data file. Recall of multi level indexes is not required.	**2**
7.1.6	Outline direct access file organization.	A direct access file can have unordered records. Students should be able to outline how records can be retrieved via a calculation followed by direct access.	**2**
7.1.7	Outline the need for fixed- and variable-length fields and records, and how they are related to direct and sequential access methods.		**2**
7.1.8	Describe the use of hash algorithms to save and retrieve records in a direct access file.	Students should understand the use of modulo operators (mod, div) in the construction of a hash function. See also 5.2.3.	**2**
7.1.9	Compare the speed of access and storage requirements for the types of files mentioned in 7.1.2–7.1.8.	This should also include storage media (disk, tape). Access speeds should be expressed in descriptions, calculations of iterations and BigO notation.	**3**
7.1.10	Explain how the logical organization of data differs from its physical organization.	For example, in a fully-indexed sequential file, records can be retrieved in alphabetical order by using the index, even though they are not stored physically in that order.	**3**
7.1.11	Outline the need for external sorts.	The sorting of files that are too large for the primary memory of a computer requires techniques based on a combination of sorting and merging. Recall of algorithms for merge sorts is not required.	**2**
7.1.12	Demonstrate an understanding of and use the different types of data streams identified in appendix 2.		**3**

THE CASE STUDY

Problem solving in computer science requires a clear description of a scenario (or context) that reflects a "real-life" problem, together with definitions of particular variables. Examinations naturally impose significant time constraints, especially when students have to read large quantities of text; nevertheless rather lengthy descriptions will be inevitable for some questions. Furthermore, the scenarios presented within examinations may be situated outside the experience of many students. This may be due to the age of the students, but also to their cultural and technological circumstances. The use of a case study should help to overcome these inequalities, while, at the same time, providing other assessment opportunities. Since the case study will be issued well in advance of the examination, it will allow students and teachers to familiarize themselves with the particular scenario and the language contained within it. Teachers will be able to collect enrichment, resource and background materials relevant to the scenario. They may wish to prepare their students in other ways, for example, by organizing visits or visiting speakers.

Case study aims

The aims of the case study are to:

- facilitate the study of a real scenario or situation involving a problem that can be solved using computer systems and that can be described fully
- exemplify the social significance and implications of computer systems
- capitalize on relatively current situations, thereby taking advantage of new initiatives or developments arising after the guide was composed
- provide a real-life situation on which to base examination questions from all sections of the syllabus
- try to reduce the variation in performance that might be created by a limited understanding of material due to the native language of the text being different from that of the student.

Format

The case study will consist of a booklet of several pages containing a variety of information. The content is likely to be mainly textual but may also contain information in the form of diagrams, flowcharts, algorithms, pictures/photographs, tables or graphs.

Procedures

The case study is constructed by the Diploma Programme computer science senior examiners every two years. The same case study will be used for both standard level and higher level examinations. The relevant number of case study booklets will be sent to schools as far in advance of the examination as possible. The same case study will be used for both examination sessions (May and November) for two years. "Clean" copies of the case study will accompany the examination papers.

Content

The case study will contain material relevant to all sections of the syllabus, both standard and higher level. Standard level students will not be asked questions based on areas of the case study that relate to higher level topics.

The examination

For both standard level and higher level, one question in paper 2 will require an understanding of the information in the case study at the appropriate depth. Students will be free to consult the case study during the examination. This structured question may also test an understanding of other topics in the syllabus and other questions in paper 2 may also refer to information in the case study but will not test the case study content in any depth.

ASSESSMENT OUTLINE

Computer science standard level

First examinations 2006

COMPONENT	WEIGHTING	OBJECTIVES (Approximate weighting)		DURATION		DETAILS AND MARK TOTALS
		1+2	3+4	Sections	Total	
External Assessment	**65%**				**3 hrs**	
Paper 1	32.5%	19%	13.5%		1 hr 30 mins	
Section A	14%	11.5%	2.5%	40 mins approx.		Several compulsory short-answer questions (30 marks)
Section B	18.5%	7.5%	11%	50 mins approx.		Four compulsory structured questions (40 marks)
Paper 2	32.5%	12%	20.5%		1 hr 30 mins	Three compulsory questions:
	18.5%	5%	13.5%	50 mins approx.		Two compulsory extended-response questions including the construction of an algorithm (40 marks)
	14%	7%	7%	40 mins approx.		One compulsory structured question based on the case study (30 marks)
Internal Assessment	**35%**					
Program Dossier	35%	20%	15%	25 hrs' teacher contact time plus further computer access time		One in-depth project addressing a single problem that enables the student to demonstrate mastery of the required aspects (50 marks)

Computer science higher level

First examinations 2006

COMPONENT	WEIGHTING	OBJECTIVES (Approximate weighting)		DURATION		DETAILS AND MARK TOTALS
		1+2	3+4	Sections	Total	
External Assessment	**65%**				**4hrs 30 mins**	
Paper 1	32.5%	19.5%	13%		2 hrs 15 mins	
Section A	13%	10.5%	2.5%	1 hr approx.		Several compulsory short-answer questions (40 marks)
Section B	19.5%	9%	10.5%	1 hr 15 mins approx.		Six compulsory structured questions (60 marks)
Paper 2	32.5%	13%	19.5%		2 hrs 15 mins	Four compulsory questions:
	19.5%	4%	15.5%	1 hr 15 mins approx.		Three compulsory extended-response questions including the construction of an algorithm (60 marks)
	13%	9%	4%	1 hr approx.		One compulsory structured question based on the case study (40 marks)
Internal Assessment	**35%**					
Program Dossier	35%	20%	15%	35 hrs' teacher contact time plus further computer access time		One in-depth project containing a single problem that enables the student to demonstrate mastery of the required aspects (50 marks)

ASSESSMENT DETAILS

External assessment

The computer science assessment model is designed to measure student performance against the four assessment **objectives**. Assessment is carried out by a combination of external examinations conducted at the end of the programme of study, and internal assessment, carried out by teachers. These two key assessment structures are respectively weighted at 65% and 35%.

Standard level

External assessment 65%

Paper 1 **(70 marks)** **32.5%**

- Paper 1 is an examination paper of **1 hour 30 minutes** consisting of **two compulsory sections**. The paper is designed to test each student's overall knowledge of the syllabus content.
- Section A (40 minutes approximately) consists of **several** compulsory short-answer questions testing mainly objectives 1 and 2. The maximum mark is 30.
- Section B (50 minutes approximately) consists of **four** compulsory structured questions testing mainly objectives 3 and 4. The maximum mark for each question is 10.

Paper 2 **(70 marks)** **32.5%**

- Paper 2 is an examination paper of **1 hour 30 minutes** consisting of **three compulsory questions**.
- The first two questions (50 minutes approximately) are extended-response questions, in several parts. They require students to construct algorithms based on appropriate scenarios. The maximum mark for each question is 20.
- The third question (40 minutes approximately) is structured, in several parts, and based on a case study. Further details about the case study can be found in this guide. The maximum mark is 30.

Calculators

The use of calculators will **not** be permitted in any computer science examinations.

Higher level

External assessment 65%

Paper 1 (100 marks) 32.5%

- Paper 1 is an examination paper of **2 hours 15 minutes** consisting of **two compulsory sections**. The paper is designed to test each student's overall knowledge of the syllabus content.
- Section A (**1 hour** approximately) consists of **several** compulsory short-answer questions testing mainly objectives 1 and 2. Several questions are common to SL paper 1 section A (20 marks approximately). The remaining questions examine HL topics. The maximum mark is 40.
- Section B (**1 hour 15 minutes** approximately) consists of **six** compulsory structured questions testing mainly objectives 3 and 4. The maximum mark for each question is 10.

Paper 2 (100 marks) 32.5%

- Paper 2 is an examination paper of **2 hours 15 minutes** consisting of **four compulsory questions**.
- The first three questions (**1 hour 15 minutes** approximately) are extended-response questions, in several parts. They require students to construct algorithms based on appropriate scenarios. The maximum mark for each question is 20.
- The fourth question (1 hour approximately) is structured, in several parts, and based on a case study (common to SL). Further details about the case study can be found in this guide. The maximum mark is 40.

Calculators

The use of calculators will **not** be permitted in any computer science examinations.

Internal assessment: program dossier (50 marks) 35%

The program dossier is an individual piece of work completed during the course. The dossier must address a **single problem** that can be solved using computer systems and which has an **identified end-user**. The analysis, design and production of the final system must be **well documented**.

The emphasis is on the use of a logical approach and analytical thinking from definition and decomposition of the problem through to its solution by constructing appropriate classes implementing algorithms and data structures in the **Java programming language**.

The program dossier is internally assessed by the teacher and externally moderated by the IBO following procedures provided in the *Vade Mecum*.

Time allocation

Standard level

It is expected that approximately 25 hours' teacher contact time will be devoted to the program dossier, including guidance on format, presentation and content. Some of the time teaching the syllabus content will also involve work connected with the program dossier, but this does not include the time required by students to work on their own to develop and complete their dossiers.

Higher level

It is expected that approximately 35 hours' teacher contact time will be devoted to the program dossier, including guidance on format, presentation and content. Some of the time teaching the syllabus content will also involve work connected with the program dossier, but this does not include the time required by students to work on their own to develop and complete their dossiers.

Choice of problem

The role of the teacher is crucial in advising the student in their choice of problem. Overly ambitious problems should be avoided as should overly simplistic ones.

Students are free to choose problems generated by themselves or their teacher, but the problem chosen must have an identified end-user. Students may share the same problem to be solved or the same initial scenario, but **collaborative work is forbidden.**

Teachers are expected to give educational guidance at each stage of the design process. In particular the prototype should be fully explored by the teacher and student to ensure that the requirements of the user can be met within the programming abilities of the student and within the time available. If this is not the case then another problem should be chosen or a restricted solution should be offered to the end-user.

The scope of mastery aspects available in the problem needs to be considered. The level of difficulty of the problem should be consistent with the ability of the student.

Approach

The internal assessment criteria imply that the work on the dossier falls into four main stages:

A—Analysis

B—Detailed design

C—The program

D—Documentation.

Preferably students will complete these stages in the order given. However, with stages B and C it may occasionally be necessary for students to return from C to B one or more times to refine their detailed design in a "spiral" of design and development. This will also depend on the nature of the problem (open or closed) and on the ability of the student. Teachers should not allow students to produce stages A and B following development of the solution.

Teachers are advised to set deadlines for the ends of stages A, B, C and D so that students are helped to achieve success.

Students are free to choose a design methodology (structured, top-down or object-oriented) that is flexible and extensible. Therefore, it may be necessary for them to retain design documentation from earlier stages to present as evidence in the final dossier in support of awards for criteria B1–B3. Teachers may wish to design their own methods of collecting such design documentation (design logs, portfolios containing CRC cards, UML style diagrams and so on). Examples will be provided with teacher support material for this course.

When the student's program is complete, the teacher should run it in the presence of the student to confirm that it functions, and has produced the hard copy output submitted with the program dossier.

Automated development systems

Some programming systems, such as visual IDEs, provide interactive development environments with a wide range of extra facilities, such as visual design, object manipulation, and automatic code generation. However, the use of these is beyond the scope of this syllabus.

Within the program dossier such facilities may be used by the student, but must not be used for mastery tasks. For example, SL students would be expected to write their own algorithms for sorting an array, rather than simply executing a library function that sorts the array. Similarly, HL students would be expected to write their own algorithms that maintain a linked data structure, rather than using a system library that already contains all the required algorithms.

Any program listing that includes code automatically generated by the development system must have this code clearly identified and distinguishable from the code written by the student.

Teacher assessment

Teachers assess students' performance by using level descriptors against the relevant criteria, which are related to the objectives. The criteria and achievement levels must be applied to the work in the program dossier **regardless** of the number of aspects in which mastery is demonstrated. After this a "mastery factor" is applied. This factor depends on the number of different aspects in which mastery is demonstrated. (See the section on mastery in this guide). The assessment of the program dossier is moderated externally.

Only the code **designed** and **written** by the student must be taken into account when applying the assessment criteria, and awarding marks.

If teachers add comments to dossiers as well as marking them ready for moderation, this facilitates the moderation process. In addition, if teachers write a report for each student that justifies the achievement level awarded for each criterion, this also will facilitate the moderation process and make the feedback forms from the moderator more focused.

Format of the dossier

All the student's work must be submitted together as a single document. The work can be stapled, put into a ring-binder or inserted into a folder. All information required for the program dossier must appear as hard copy. Diskettes, CD-Roms, and so on must **not** be included within the program dossier or sent to the moderator.

There must be a table of contents and all written documentation should be word processed, except where it is felt necessary to include rough notes. Program runs and sample screens may be annotated by hand.

All the pages must be numbered. The numbering can be sequential (1, 2, 3, and so on) throughout the entire program dossier or it can be done according to the items numbered in the following table. (For example, if the design process is the third item, then these pages can be numbered 3–1, 3–2, 3–3, and so on.) This may be easier, since each item can be numbered sequentially as it is completed. The page numbering can be done by hand if the available computer systems do not permit automatic page numbering.

The number of pages associated with each item may vary according to the nature and complexity of the problem being solved as well as its programmed solution. However, as a guide, an approximate number of pages is given in the following table. This is included mainly to ensure that students include the appropriate amount of material.

Items to be included in the program dossier

All of the items listed in the following table must be included in the program dossier.

Items to be included in the program dossier	Suggested number of pages
Table of contents	
Analysis of the problem	2–3
Criteria for success	1–2
Prototype solution	Variable
Data structures	2–5
Algorithms	2–5
Modular organization	3–5
Usability	1
Handling errors	1–2
Code listing	Variable (500–3,000 lines)
Annotated hard copy	Variable
Evaluation of solutions	2
User documentation	6
Documentation of mastery aspects	2
Total	Approx 60–100

Internal assessment criteria

Using the assessment criteria and descriptors

The method of assessment used by the IBO is criterion related. That is to say each student is assessed against identified assessment criteria and not against other students.

- There are **fourteen** assessment criteria for the program dossier. For each assessment criterion, achievement level descriptors are defined that concentrate on positive achievement, although for the lower levels (1 = the lowest level of achievement) failure to achieve may be included in the description.
- The aim is to find, for each criterion, the descriptor that conveys most adequately the achievement level attained by the student.
- Having scrutinized the work to be assessed, read the descriptors for each criterion, starting with level 1, until you reach one that describes a level of achievement that the work being assessed has **not** reached. The work is therefore best described by the preceding achievement level descriptor and you should record this level.
- Use only whole numbers, not partial marks such as fractions and decimals. If a student does not achieve a standard described by any of the descriptors, then 0 (zero) should be recorded.
- The highest descriptors do not imply faultless performance and teachers should not hesitate to use the extremes, including zero, if they are appropriate descriptions of the work being assessed.
- Descriptors should not be considered as marks or percentages, although the descriptor levels are ultimately added together to obtain a score out of 50. It should not be assumed that there are other arithmetical relationships; for example, a level 4 performance is not necessarily twice as good as a level 2 performance.
- A student who attains a particular level of achievement in relation to one criterion will not necessarily attain similar levels of achievement in relation to the others. Do not assume that the overall assessment of the students will produce any particular distribution of scores.

Stage A—Analysis

Criterion A1: Analysing the problem

The documentation should be completed first and contain a thorough discussion of the problem that is being solved. This should concentrate on the **problem** and the goals that are being set, not on the method of solution. A good analysis includes information such as sample data, information and requests from the **identified end-user**, and possibly some background of how the problem has been solved in the past. A **systematic method** is one that takes into account what input and output will occur and what calculations and processes will be necessary to obtain the desired output.

0	The student has not reached a standard described by any of the descriptors given below. For example, the student has simply described the programmed solution.
1	The student only **states** the problem to be solved **or shows** some evidence that relevant information has been collected.
2	The student **describes** the problem to be solved.
3	The student describes the problem **and provides evidence** that information relating to the problem has been collected.
4	The student provides evidence that a **systematic method** has been used in the analysis of the problem.

This section of the program dossier would typically be two to three pages in length. It should include a brief statement of the problem as seen by the end-user. A discussion of the problem from the end-user's point of view should take place, including the user's needs, required input and required output. For example, evidence could be sample data, interviews and so on, and could be placed in an appendix.

Criterion A2: Criteria for success

This section of the program dossier will clearly state the objectives/goals of the solution to the problem. The expected behaviour of the solution should be clearly described and the limits under which it can operate outlined.

0	The student has not reached a standard described by any of the descriptors given below.
1	The student **states some** objectives of the solution.
2	The student **describes most of** the objectives of the solution.
3	The student **relates all of** the objectives of the solution to the analysis of the problem.
4	The student relates all of the objectives of the solution to the analysis of the problem, and **outlines** the limits under which the solution will operate.

This section of the program dossier would typically be one to two pages in length. Objectives should include minimum performance and usability. These criteria for success will be referred to in subsequent criteria, for example criterion C2 (Usability), C4 (Success of program); D2 (Evaluating solutions) and D3 (Including user documentation).

The limits under which the solution will operate will vary. Some examples are:

- Time taken to return a research result from a data file
- The response of the program to invalid and extreme data input
- Limitations on the volume of data stored in the program
- Usability of user input screen
- The proper response of the program to user input.

Criterion A3: Prototype solution

The prototype solution **must** be preceded by an initial design for some of the main objectives that were determined to be the criteria for success. A prototype of the solution should be created.

A prototype is: "The construction of a simple version of the solution that is used as part of the design process to demonstrate how the system will work.

The prototype need not be functional, it could be constructed using a number of tools such as: Visual Basic, PowerPoint, Mac Paint, Corel Draw for a simple Java program. The intent is to show the user how the system is expected to operate, what inputs are required and what outputs will be produced. A number of screenshots will be required for the user to be able to evaluate the solution properly. The prototype, at its simplest, could be a series of clear, computer-generated drawings, a hierarchical outline of features in text mode, or a series of screenshots.

Documentation of user feedback could be, for example, a report of the user's comments on the prototype.

Stage B—Detailed design

The ordering of the criteria B1–B3 does not imply that this is the sequence in which students should develop or document their designs. This will vary according to the methodology adopted.

Criterion B1: Data structures

Students should choose data structures, **at the design stage**, that fully support the data-storage requirements of the problem, and that allow clear, efficient algorithms to be written. The data structures must fully support the objectives of the solution (criterion A2). The classes chosen should be logical in that the data is sensible for the objects in question and the methods are appropriate for the data given. This section of the program dossier could include class definitions, file structures, abstract data types (particularly at higher level) and some consideration of alternatives.

0	The student has not reached a standard described by any of the descriptors given below.
1	The student has **outlined some** of the data structures/types to be used in the solution.
2	The student has **described some** of the data structures/types to be used, and provided sample data.
3	The student has **discussed all** of the data structures/types to be used, and provided sample data.
4	The student has discussed and **clearly illustrated all** of the data structures/types to be used to solve the problem, and provided sample data for **all** of them.

This section would typically be two to five pages in length.

Data structures and data members that are to be used in the programmed solution should be discussed here. Sample data, sketches/illustrations, including discussion of the way data objects will be changed during program execution should be demonstrated to achieve a level 4 in criterion B1.

Criterion B2: Algorithms

Students should choose algorithms, **at the design stage**, that fully support the processes needed to achieve the objectives of the solution (criterion A2), and provide sufficient support for the required data structures. The classes chosen should be logical in that the methods are appropriate for the data given. Students must include parameters, return values, and descriptions of pre- and post-conditions.

0	The student has not reached a standard described by any of the descriptors given below.
1	The student has **outlined some** of the algorithms to be used in the solution.
2	The student has **described most** of the algorithms to be used, with details of parameters and return values.
3	The student has **discussed all** of the algorithms to be used, with details of parameters, return values, pre-conditions and post-conditions.
4	The algorithms discussed are sufficiently **logical**, **detailed**, and **well documented** to be used to create the solution in Java.

This section would typically be two to five pages in length.

This can be a list or outline of all the algorithms, presented as text, possibly in outline format. Standard algorithms (such as search or sort) can simply be named (with parameters), but non-standard algorithms must be described in more detail.

Criterion B3: Modular organization

Students should choose modules, **at the design stage**, that incorporate the data structures and methods required for the solution (criteria B1 and B2) in a logical way. The data structures must fully support the objectives of the solution (criterion A2). Students must present this organization in a structured way that clearly shows connections between modules (hierarchical decomposition or class dependencies). The connections between modules, algorithms and data structures must also be presented.

0	The student has not reached a standard described by any of the descriptors given below.
1	The student has **outlined some** of the modules to be used in the solution.
2	The student has **described most** of the modules to be used, **showing connections** between them.
3	The student has **described all** of the modules to be used, and has shown the connections to data structures and methods.
4	The modules discussed are sufficiently **logical, detailed** and **well documented** to be used to create the solution in Java.

This section would typically be three to five pages in length.

A variety of presentations are possible here. Some possibilities are:

- a top-down hierarchical decomposition chart containing the names of modules, showing connections between modules and showing details of which data structures and methods are connected with (or part of) which modules
- a text outline showing hierarchical decomposition (equivalent to above)
- a hard copy of CRC cards showing dependencies between collaborating classes, with details of which data structures and methods are connected with (or part of) which classes.

The design is assessed independently from the programming stage (stage C). The design should be complete, logical and usable, but the student may deviate from it or expand it during stage C, without penalty.

Stage C—The program

Program listings must contain **all** the code written by students and, if a program listing displays code that was automatically generated by the development system or copied from another source, then this code must be clearly identified and distinguishable from that code written by the students. Only the code **designed and written** by students must be taken into account when applying the assessment criteria.

Criterion C1: Using good programming style

Good programming style can be demonstrated by program listings that are easily readable, even by a programmer who has never used the program. These would include small and clearly structured Java methods, sufficient and appropriate comments, meaningful identifier names and a consistent indentation scheme.

0	The student has not reached a standard described by any of the descriptors given below.
1	The program listing demonstrates **some** attention to good programming style.
2	The program listing **mostly** demonstrates attention to good programming style.
3	All parts of the program listing demonstrate **considerable** attention to good programming style.

A typical program should be approximately 1,000–3,000 (HL) or 500–2,000 (SL) lines of code in length.

Comments should be included to describe the purpose and parameters of each method, and also when code is difficult to understand.

The program should demonstrate the use of good programming techniques. It should include:

- an identification header indicating the program name
- author, date, school
- computer used, IDE used, purpose.

The program should possess good internal documentation, including:

- constant, type and variable declarations that should have explanatory comments
- identifiers with meaningful names
- objects that are clearly separated and have comments for their parameters
- suitable indentation that illustrates various programming constructs.

Generally, achievement level 2 will be appropriate where two or more of these have been demonstrated. Then, achievement level 3 will be appropriate for three or more being demonstrated.

Criterion C2: Usability

Students should pay attention to issues of **usability** during the design stage. To be given credit students must include features that make the program more user-friendly, such as helpful menus, help instructions, useful guidance to the user during the execution of the program.

The features should go beyond the basic requirements needed to operate the program.

This criterion does not refer to internal error checking.

0	The student has not reached a standard described by any of the descriptors given below.
1	The student documents **some** user-friendly features within the program.
2	The student **fully** documents the user-friendly features of the program.
3	The student **fully** documents the user-friendly features of the program, and the program **meets** the usability objectives in criterion A2.

This section would typically be no more than one page.

Separate output is not required for this section. However, a usability section should be present that summarizes the features, making reference to the annotated sample runs and the program listing. Features could be annotated, commented or highlighted in those sections and a simple table presented here.

The teacher should run the program with the student before awarding achievement levels for this criterion.

Criterion C3: Handling errors

This refers to detecting and rejecting erroneous data input from the user, and preventing common run-time errors caused by calculations and data-file errors. Students are not expected to detect or correct intermittent or fatal hardware errors such as paper-out signals from the printer or damaged disk drives, or to prevent data-loss during a power outage.

0	The student has not reached a standard described by any of the descriptors given below.
1	The student **includes** documentation that shows a **few** error-handling facilities in the program, or documents only **one** type of input or output.
2	The student includes documentation that shows **many** error-handling facilities in the program, and documents **more than one** type of input or output.
3	The student **fully** documents the error-handling of **each** input and output method within the program.

This section would typically be one to two pages in length.

For this criterion, students must attempt to trap as many errors as possible. The documentation in the dossier can take a variety of forms.

For example, students could highlight relevant comments within the program listing or they could produce a table with two columns, one that identifies any error possibilities, and one that shows the steps taken to trap the errors. It is not expected that extra output is produced for this section.

Criterion C4: Success of the program

Evidence here refers to hard copy output in criterion D1.

0	The student has not reached a standard described by any of the descriptors given below.
1	The student **includes** evidence that the program **functions partially**. The student successfully achieved **some** of the objectives from criterion A2.
2	The student includes evidence that the program functions **well**. The student successfully achieved **most** of the objectives from criterion A2.
3	The student includes evidence that the program functions well. The student successfully achieved **all** of the objectives from criterion A2.

The teacher should run the program with the student to confirm that the program functions, and that it produces the hard copy output submitted with the program dossier.

Stage D—Documentation

Criterion D1: Including an annotated hard copy of the test output

The hard copy of test output should demonstrate that the program fulfills the criteria for success in criterion A2. The output **must** be **annotated** (this may be done by hand). The teacher must confirm that each student has actually completed the testing as claimed in the documentation. (See the *Vade Mecum*.)

0	The student has not reached a standard described by any of the descriptors given below.
1	The student includes an **incomplete** set of sample output.
2	The student includes an incomplete set of **annotated** sample output.
3	The student includes a **mostly complete** set of annotated sample output.
4	The student includes a **complete** set of annotated sample output, testing all the objectives in criterion A2.

Hard copy output from one or more sample runs should be included to show that the different branches of the program have been tested; testing one set of valid data will not be sufficient. The hard copy submitted should demonstrate the program's responses to inappropriate or erroneous data, as well as to valid data. Thus the usefulness of the error-handling routines mentioned above should become evident. While at least one complete test run must be included in the dossier, it is not necessary that the hard copy reflect every key stroke of every test run. Cutting and pasting of additional test runs should be done to illustrate the testing of different aspects of the program.

All test runs should be annotated in such a way that the student is stating what aspect of the program is being tested. Sample output must **never** be altered by hand, erased or covered up.

Sample output can be "captured" and combined electronically with explanatory annotations into a single document. However, it is forbidden to alter or reformat sample output in any fashion (except to add page numbers or annotate in order to highlight user friendliness or error-handling facilities as discussed above), especially if these alterations would give an unrealistic impression of the performance of the program. Examples of such "abuse" include: lining up text that was not originally aligned; adding colour or other special effects; changing incorrect numerical output; erasing evidence of errors.

Criterion D2: Evaluating solutions

The evaluation/conclusion section should be a critical analysis of the resulting solution. Effectiveness should be discussed in relation to the original description of the problem and the criteria for success that were stated in criterion A2. Efficiency may be discussed in general terms, for example BigO notation is not required. Suggested improvements and possible extensions should be realistic, for example suggestions should not include statements such as "the program would be a lot better if it incorporated some artificial intelligence techniques such as speech recognition and natural language parsing".

0	The student has not reached a standard described by any of the descriptors given below.
1	The student only **outlines** the solution.
2	The student **outlines** the solution and **partly** considers effectiveness, efficiency and possible improvements.
3	The student **discusses** the effectiveness and efficiency of the solution **and suggests** alternative processes and improvements.
4	The student **suggests** alternative approaches to the solution and the design process.

This section of the dossier would typically be two pages in length.

The evaluation/conclusion should include reflections on the effectiveness of the programmed solution of the original problem. It should discuss answers to the following questions.

- Did it work?
- Did it address the criteria for success?
- Did it work for some data sets, but not others?
- Does the program in its current form have any limitations?
- What additional features could the program have?
- Was the initial design appropriate?

A thorough evaluation should also discuss possible future enhancements that could be made to the program.

Criterion D3: Including user documentation

Good documentation usually includes both sample output and written instructions. It should be sufficiently complete that it will allow anyone unfamiliar with the program to start using it effectively after reading the instructions. The documentation must include some screenshots to illustrate program operation. These screenshots should be separate from the hard copy in criterion D1. The user documentation must be presented as hard copy.

0	The student has not reached a standard described by any of the descriptors given below.
1	The student includes **some** user documentation.
2	The student includes user documentation that contains **clear** instructions about **running** the program.
3	The student includes user documentation that contains clear, **illustrated** instructions about **installing** and running the program.

This section would typically be six pages in length.

If screenshots cannot be produced then a text version may be substituted.

Illustrated instructions about installing and running the program on another machine without using an IDE should be provided for achievement level 3 in criterion D3. For example, compiled classes can be copied to another computer and run.

Stage E—Holistic approach

Criterion E: Holistic approach to the dossier

The program dossier should be an ongoing process involving consultation between the student and teacher. The student should be aware of the expectations of the teacher from the beginning of the process and each achievement level awarded should be justified by a written comment from the teacher at the time of marking. The examples given below for each criterion level are teacher-orientated and each teacher should use discretion when judging the levels.

0	The student showed **no** commitment. For example, the student did not participate in class discussions on dossier work, did not submit the required work in progress, and/or missed many deadlines.
1	The student showed **minimal** commitment. For example, the student participated minimally in class discussions on dossier work, kept to most deadlines, had some discussion initiated by the teacher and/or did not exploit the available opportunities for the development or improvement of the dossier.
2	The student showed **good** commitment. For example, the student participated in class discussions on dossier work, initiated discussions with the teacher and/or the rest of the class and/or became fully involved in the development of the dossier.
3	The student showed **full** commitment. For example, the student participated fully in class discussions on dossier work, took initiatives both in discussion with the teacher and/or the rest of the class and in subsequent work of a more independent nature and/or demonstrated a full understanding of all the steps in the development of his/her dossier.

In order to obtain the highest achievement level for this criterion the student should have excelled in areas such as those listed below. This list is not exhaustive and teachers are encouraged to add their own expectations.

The student:

- actively participated at all stages of the development of the dossier
- demonstrated a full understanding of the concepts associated with his/her dossier
- demonstrated initiative
- demonstrated perseverance
- showed insight
- prepared well to meet deadlines set by the teacher.

MASTERY

Students must demonstrate mastery of various aspects of Java by documenting evidence in their program dossiers.

Mastery aspects

Standard level

To achieve a mastery factor of 1.0, students must have mastered at least **10** of the following 15 aspects.

1. Arrays
2. User-defined objects
3. Objects as data records
4. Simple selection (**if–else**)
5. Complex selection (nested **if, if** with multiple conditions or **switch**)
6. Loops
7. Nested loops
8. User-defined methods
9. User-defined methods with parameters (the parameters have to be useful and used within the method body)
10. User-defined methods with appropriate return values (primitives or objects)
11. Sorting
12. Searching
13. File i/o
14. Use of additional libraries (such as utilities and graphical libraries **not included** in appendix 2 Java Examination Tool Subsets)
15. Use of sentinels or flags

It is anticipated that this list will provide students with the option to choose algorithms and data structures appropriate to the problem rather than contriving a solution to fit the mastery aspects.

Where one aspect includes others, all are credited, for example aspect 10 will also satisfy aspects 8 and 9 (always provided that the use is **non-trivial**, **well-documented** and **appropriate**).

Higher level

To achieve a mastery factor of 1.0, students must have mastered at least **10** of the following 15 aspects.

1. Adding data to an instance of the **RandomAccessFile** class by direct manipulation of the file pointer using the **seek** method
2. Deleting data from an instance of the **RandomAccessFile** class by direct manipulation of the file pointer using the **seek** method. (Data primitives or objects may be shuffled or marked as deleted by use of a flag field. Therefore files may be ordered or unordered).
3. Searching for specified data in an instance of the **RandomAccessFile** class.
4. Recursion
5. Merging two or more sorted data structures
6. Polymorphism
7. Inheritance
8. Encapsulation
9. Parsing a text file or other data stream
10. Implementing a hierarchical composite data structure. A composite data structure in this definition is a class implementing a record style data structure. A hierarchical composite data structure is one that contains more than one element and at least one of the elements is a composite data structure. Examples are, an array or linked list of records, a record that has one field that is another record, or an array.
11. The use of any five standard level mastery factors—this can be applied only once

12–15. Up to four aspects can be awarded for the implementation of abstract data types (ADTs) according to the table entitled "Implementation of ADTs".

An ADT may be implemented as a class or interface containing data members and methods appropriate to that ADT. The number of mastery aspects to be awarded will depend on the thoroughness and correctness of the student's implementation. Examples are given in the following table.

"Non-trivial" means that the programmer must demonstrate that the program benefits from the use of the aspect.

Where one aspect includes others, all are credited (always provided that the use is **non-trivial, well documented** and **appropriate**).

Implementation of ADTs

ADT name	One aspect	Two aspects	Three aspects	Four aspects
General criteria.	An incomplete ADT is implemented.	An ADT is implemented with all key methods implemented.	An ADT is implemented that includes some error checking.	An ADT is implemented completely and robustly.
Lists, implemented using references (that is, a dynamically linked list).	A node style class with appropriate constructors and methods to set and get data elements.	Methods are implemented to add at/remove from the tail and the head of the list.	Proper checks are made for errors such as attempting to get an element from an empty list or inserting the same element twice.	All error conditions are checked for, and all appropriate methods are implemented. For a doubly linked list these could be: size isEmpty first last before after insertHead insertTail insertAfter insertBefore.
Tree (simple, ordered, binary tree is sufficient using arrays or dynamically linked object instances).	A class or interface with appropriate constructors and methods to set and get data elements.	Methods are implemented to add at/remove from the correct point in the tree.	Proper checks are made for errors such as attempting to get an element from an empty tree or not inserting the same element twice.	All error conditions are checked for, and all appropriate methods are implemented. For a simple, ordered, binary tree these could be: size isEmpty root parent leftChild rightChild.

ADT name	One aspect	Two aspects	Three aspects	Four aspects
Stack implemented dynamically or statically.	A class or interface with appropriate constructors and methods to push and pop items.	Methods to test for full and empty stack are added.	Proper checks are made for errors such as attempting to get an element from an empty stack.	Probable methods: push pop top isEmpty isFull size.
Queue implemented dynamically or statically.	A class or interface with appropriate constructors and methods to enqueue and dequeue items.	Methods to test for full and empty queue are added.	Proper checks are made for errors such as attempting to get an element from an empty queue.	Probable methods: enqueue dequeue front rear isEmpty isFull size.
Hash table implemented in an array.	A class or interface with appropriate constructors and methods to insert and remove items.	Methods to test for full table and duplicate keys are added.	Proper checks are made for errors such as attempting to get a non-existent key, clashes are dealt with properly.	Probable methods: hashFunction insertKey removeKey isDuplicate isEmpty isFull size.

It is not possible to provide an exhaustive list here and teachers will need to exercise some degree of judgment in implementing this mastery aspect. It is considered highly unlikely that teachers will encourage students to develop graphs, heaps, dictionaries, priority queues and ADTs of similar complexity.

“Complete and robust” means that all the needs of the solution are solved without failure.

The mastery factor

Mastery judgments must be made in the same way for both SL and HL. Therefore the criteria should be applied in the same way to the work in both SL and HL program dossiers.

Both SL and HL students are required to demonstrate mastery of **at least 10** aspects. The criteria and level descriptors should first be applied to the work in the dossier **regardless** of the mastery aspects demonstrated.

The appropriate mastery factor should then be determined from the table below. After applying the mastery factor, the student's final score should be rounded to the nearest whole number (0.5 or above rounds up to the next whole number).

Students must also document their dossiers thoroughly. To show mastery of an aspect, it is **not** sufficient for students only to **use** it within a program. In the written documentation, students must include information about **why** a particular data structure is appropriate, **how** it is used (for example, how nodes are added, deleted and searched for) and **where** it is used in the program. In other words, students must provide cross-references between the documentation and specific procedures within the program.

Number of aspects in which the student demonstrates mastery	Mastery factor
10 or more	1.0
9	0.9
8	0.8
7	0.7
6	0.6
5	0.5
4	0.4
3	0.3
0, 1 or 2	0.2

Examples

1. A student achieves 29, as judged by applying the assessment criteria.

   Mastery was demonstrated in eight different aspects.

   So the final mark awarded is $29 \times 0.8 = 23.2 = 23$.
2. A student achieves 32, as judged by applying the assessment criteria.

   Mastery was demonstrated in twelve different aspects.

   So the final mark awarded is $32 \times 1.0 = 32$.

Documentation of mastery aspects

The mastery aspects should be listed with:

- the page number(s) where they occur in the code listing
- a brief description of how their use benefits the solution.

APPENDIX 1

Glossary of computer science terms

No list of computer science terms can be exhaustive. This glossary includes terms relevant to the IB Diploma Programme computer science course and these are not necessarily applicable universally. Texts do not always agree about the definitions of some terms, but ambiguity should be reduced in cases where more than one word is used for the same concept by using the definition given in the glossary.

Terms that are relevant for the higher level (HL) course only are indicated by HL in the second column.

abstract data structure	HL	A way of organizing data and its related procedures and functions.
accessor methods	HL	Methods that do not alter the state or attributes of an object; their purpose is to return information.
accumulator	HL	A storage register in the ALU that holds data temporarily while the data is processed and before it is transferred to memory.
A–D converter		Analog–digital converter. A device for converting analog signals into digital ones for subsequent computer processing; sometimes called a "digitizer". A digital to analog (D to A) converter operates in the reverse direction.
ADSL (Asymmetrical Digital Subscriber Line)	HL	Technology that increases the data rate over existing telephone lines accommodating voice and digital data transfer. A special modem is needed for access.
address bus		Pathway from memory to processing unit that carries the address in memory to and from which data is transferred. See the definitions for "bus" and "data bus".
algorithm		An ordered set of well-defined instructions for the solution of a problem in a finite number of steps.
ALU		See the definition for "arithmetic and logic unit".
analog data		The representation and measurement of the performance or behaviour of a system by continuously variable physical entities such as currents, voltages and so on. See also the definition for "digital data".

and		The output of "and" is True if all statements are True, False if any statement is False.
applet (Java)		A program that runs in the context of a browser.
application (Java)		A program that runs when translated by a Java compiler.
archive		Data that represents a record of data held and processed at a specific time, which is held off-line for future research or for legal reasons.
argument	HL	A value or object passed to a method when it is called.
arithmetic and logic unit (ALU)		A part of the computer that performs arithmetic operations, logic operations and related operations.
array		1. An arrangement of data in one or more dimensions. 2. In programming languages, an aggregate that consists of data objects, with identical attributes, each of which may be uniquely referenced by indexing.
ASCII: American Standard Code for Information Interchange		The primary encoding character set used in computers for textual data transfer between applications. The set uses eight bits for each character code, one of these bits being a check bit to verify the seven bits needed to represent one character. ASCII supports most European alphabets. Unicode supports most known alphabets and is increasingly used in data transfer. See also the definition for "Unicode".
attribute	HL	Element of data contained in an object; as specified within the object's class.
B		Byte.
back-up (file)		A second copy of a file, to be used in the event of the original file being corrupted.
balanced tree	HL	A tree in which the right and left subtrees of any node have heights differing by one at the most. See also the definition for "unbalanced tree".
bar code		A pattern of vertical lines distinguished from each other by width. It can be read by a bar code reader to provide data to a computer.
bar code reader		An optical reader that can read bar codes.

base		The basis of a notation or number system, defining a number representational system by positional representation. In a decimal system the base is 10, in a hexadecimal system the base is 16, and in a binary system the base is 2.
batch processing		A method of processing data in which transactions are collected and prepared for input to the computer for processing as a single unit, for example, payroll.
behaviour	HL	The way in which an object reacts to the methods applied
BigO notation	HL	A notation used to describe the relative performance (speed) of an algorithm.
binary operator	HL	An operator that combines two operands to give a single result, for example, addition, multiplication, division, mod, div. See also the definition for "unary operator".
binary search		A search in which, at each step of the search, the set of data elements is divided by two, until the searched element is found. See also the definition for "sequential search".
binary tree	HL	A tree in which each node has at most two children.
bit (b)		Binary digit. The smallest unit of information for data storage and transmission. Each bit is considered to be either a "0" or a "1".
block		The smallest unit of data that can be transferred between memory and backing store in one operation.
BMP		An extension given to files in bitmap form.
Boolean expression		An expression that has a value of True (T) or False (F).
bps		Bits per second.
browser		Generally used to give interactive access to information on the World Wide Web, retrieving web pages and displaying in a multi-media format.
bubble sort		A sort in which the first two items to be sorted are examined and exchanged if necessary to place them in the specified order; the second item is then compared with the third (exchanging them if required), the third is compared with the fourth, and the process is repeated until all pairs have been examined and all items are in the proper sequence. See also the definitions for "insertion sort", "selection sort" and "quicksort".

buffer		A portion of storage used to hold input or output data temporarily.
bus		The pathway used for sending signals between internal components of a computer. Components can share the same bus but cannot transmit simultaneously. See also definitions of "data bus" and "address bus".
bus topology		A network in which all devices are connected to a common cable, known as the "bus". See also definitions of "star topology" and "tree topology".
Byte (B)		A set of bits considered as a unit; it normally consists of 8 bits and corresponds to a single character of information.
cable		Wire or glass fibre used to connect computers over a network. Copper (coaxial and twisted pair) and glass fibre (fibre optic cable) are the most common.
cache		Part of the main store that is between main memory and the processor. It holds a copy of data and instructions that are likely to be used next by the processor and is hence faster than main memory. See also the definition for "disk cache".
CASE		See the definition for "computer-assisted software engineering".
character set		A finite set of different characters that is complete for a given purpose, for example, the 128 ASCII characters.
check digit		A digit added to numerical data that can be recalculated and hence used to check data integrity after input, transmission and so on.
check sum		A sum generated using individual digits of a number and employed as an error-detecting device.
circular queue	HL	A queue in which the storage area is fixed and the first item is held in a location that is logically next to the storage location for the last item of the queue. Data items can be thought of as being arranged in a circle.
clash (collision)	HL	A situation in which two or more entries in a file or other data structure are given the same memory location through the use of a hash table.
class		Combination of data and operations that can be performed on that data; specification of the data members and methods of the object.

client		Desktop computer or terminal used to access a computer-based system.
client–server		A network architecture in which a system is divided between server tasks performed on the instructions received from clients, requesting information.
collection		A class designed to hold objects (referred to in the syllabus as data structure).
command language	HL	A set of procedural operators with a related syntax, used to indicate the functions to be performed by an operating system.
compiler		A program that translates a source program into machine code that can be converted into an executable program (an object program). See also the definition for “interpreter”.
computer-assisted software engineering		The automation of well-defined methodologies that are used in the development and maintenance of products. These methodologies apply to nearly every process or activity of a product development cycle, for example: project planning, product designing, coding and testing.
computer architecture		The logical structure and functional characteristics of a computer, including the interrelationships among its hardware and software components.
computer program		A sequence of instructions suitable for processing by a computer.
constructor method		A method with the same name as the class that initializes the instance variables of an object of the class when the object is instantiated.
CRC cards		Class, responsibility, collaboration cards. A design tool for classes that lists a class’s name, its responsibilities and the classes with which it collaborates on an index card.
cylinder	HL	Concentric disk tracks of a hard disk (one on top of the other) form a cylinder.
database management system (DBMS)		A computer-based system for defining, creating, manipulating, controlling, managing and using databases.
data bus		The pathway between the memory or peripheral and processing unit that carries data for processing or data that has been processed. See also definitions for “bus” and “address bus”.

data compression		A method of reducing the size of data. All redundancy in the data is removed to reduce the storage needed or to speed up transfer. The data can be uncompressed back to its original state.
data integrity		The correctness of data after processing, storage or transmission.
data member		A data type that is a member of a class.
data packet		Part of a transmitted message that is sent separately. Apart from containing a portion of the message it will have other data such as check digits, destination address and so on.
data protection		Method of ensuring that personal data is correct and is not misused either by those holding it or others who have no right to access it.
data security		Method of ensuring that data is correct, safe and cannot be read or changed by those who have no right to access it.
DBMS		See the definition for "database management system".
debugging tool		A program used to detect, trace and eliminate errors in computer programs or other software.
defragmentation software		An application that reads file segments from non-contiguous sections of a storage device and then writes the files to the same device in such a way that each file segment is contiguous.
De Morgan's law	HL	If A and B are Boolean expressions, then $\overline{A + B} = \overline{A} \cdot \overline{B}$ $\overline{A \cdot B} = \overline{A} + \overline{B}$
dequeue	HL	To remove an item from the front of a queue. See also the definition for "enqueue".
digital data		Discrete data.
digital signature		A digital code attached to an electronic message or document, which is unique and which can be used to authenticate the sender or owner. Most often used in electronic commerce.
direct access file		A file organized in such a way that a calculation provides the address (location) of a record so that the record can be accessed directly. The records in the file may be ordered or unordered.

DMA	HL	Access to memory and devices without the direct control of the processor. This is most often used for hard disk access and screen display.
disk cache		RAM set aside to speed up access to a hard drive. This may be part of the disk itself or may be incorporated in cache memory.
distributed processing		A network in which some or all of the processing, storage and control functions, in addition to input/output functions, are dispersed among its nodes.
double buffering	HL	Two areas of memory set aside for data transfer between the processor and peripherals. As one is emptied the other is filled up in order to speed up transfer.
doubly linked list	HL	A linked list in which each node has both a head pointer and a tail pointer.
dynamic data structure	HL	Data structures that can change in size during program execution. See also the definition for “static data structures”.
encapsulation	HL	The combination of data and the operations that act on the data to form a single program unit called an “object”.
encryption		In computer security, the process of transforming data into an unintelligible form in such a way that the original data cannot be easily obtained except by using a decryption process.
enqueue	HL	To add an item to the rear of a queue. See also the definition for “dequeue”.
exception		An object that is created when an abnormal situation arises in a program. See also the definition for “exception handler”.
exception handler		A program code that handles exceptions that arise during the running of a program. An exception is thrown to the handler rather than causing a fatal error. See also the definition for “exception”.
expression		A sequence of symbols that can be evaluated.
fibre optic		Cabling used for networking that uses fine strands of glass. The medium can carry a great deal of data and it gives a fast transfer rate.
field (object attribute)		A subdivision of a record containing a unit of information. For example, a payroll record might have the following fields: clock number, gross pay, deductions and net pay.

FIFO	HL	First-in-first-out. See also the definitions for “queue”, “stack” and “LIFO”.
file		An organized collection of data.
file manager		An application software that can access, create, modify, store and retrieve files.
fixed-length records	HL	Records whose size is determined in advance. All such records in a file have the same length. See also the definition for “variable-length records”.
fixed point	HL	The performing of arithmetical calculations without regard to the position of the radix point. The relative position of the point has to be controlled during calculations.
flag		An indicator with two possible states, “set” or “not set”, that can be represented by one bit. A flag can be used to indicate that a record can be deleted, to indicate end of input/output and to sense whether an interrupt has occurred.
floating point	HL	In floating point arithmetic, the position of the decimal point does not depend on the relative position of the digits in the numbers (as in fixed point arithmetic), since the two parts of the floating point number determine the absolute value of the number.
formal parameter		See the definition for “parameter”.
formatted output		Data prepared for output in order to be displayed in a desired format (for example, trailing zero on 7.50$ instead of 7.5$).
fully-indexed file	HL	A file in which, although the records are unordered, a particular record can be found using a sequential access to the index of the file followed by direct access to the data file. See also the definition for “partially-indexed file”.
gateway	HL	A link between two computer systems that converts data passing through into the formats needed for each system.
graphics tablet (graphics pad)		An input device on which the user writes or designs. The image is reproduced on the screen.
GUI		Graphical user interface.
hacking		Obtaining unauthorized access to protected resources.

handshaking	HL	The exchange of predetermined signals when a connection is established between two devices or components.
hash code	HL	A method of coding to obtain a search key for the purpose of storing and retrieving items of data.
hash table	HL	A table of information that is accessed by way of a shortened search key (the hash value).
hexadecimal		A system of numbers with the base 16; hexadecimal digits range from 0 to 9 and from A to F, where A represents 10 and F represents 15.
high-level language		A programming language whose concepts and structures are convenient for human reasoning. Such languages are independent of the structures of computers and operating systems.
HTML (Hyper Text Markup Language)		A computer language used to construct web pages. Tags are used to denote the way in which text and graphics are to be displayed. The language is interpreted by a browser to display the pages.
hub		In networking, a switch that sends data to the stations to which it is attached.
IDE (integrated development environment)		A programming tool that gives programmers a single environment (that is, the hardware and software environment in which the program runs) for building programs rather than using individual editors and debuggers.
identifier		The name or label chosen by the programmer to represent a variable, method, class, data type or any other element defined within the program.
infix notation	HL	A notation for representing logical operators in which the operator is written between the operands, for example, A+B or A*B. See also the definitions for “postfix notation” and “prefix notation”.
inheritance	HL	The name given to the property whereby an object, which extends another object, inherits the data members and member functions of the original.
in-order traversal	HL	Traversal of a tree visiting the nodes in the order left-child, parent, right-child. See also the definitions for “pre-order traversal” and “post-order traversal”.
insertion sort	HL	A sort in which each item in a set is inserted into its proper position in the sorted set according to a specified criterion. See also the definitions for “bubble sort”, “selection sort” and “quicksort”.

interface		The hardware and associated software needed for communication between processors and peripheral devices to compensate for the difference in their operating characteristics.
interpreter		A program that translates and executes each instruction of a programming language before it translates and executes the next instruction. See also the definition for "compiler".
interrupt	HL	A suspension of a process, such as the execution of a computer program caused by an external event, performed in such a way that the process can be resumed.
ISDN (integrated services digital network)	HL	An international communications standard for sending voice, video and other data over digital telephone lines.
ISO		International Organization for Standardization.
iteration		The process of repeatedly running a set of computer instructions until some condition is satisfied.
JPEG (joint photographic expert group)		A recognized standard of compression of graphics files that has some loss.
keys	HL	1. In computer security, a sequence of symbols used with a cryptographic algorithm for encrypting or decrypting data. 2. In databases, the key of a record is a field with a unique value that can be used to locate that record.
latency		See the definition for "rotational delay".
left-child	HL	In a tree, the node to the immediate left of a parent node. See also the definitions for "parent" and "right-child".
library manager	HL	Many programming languages permit user-defined functions to be stored centrally and re-used in various programs. This central storage is called a "library". A library manager is a utility program that catalogues, pre-compiles and links library modules.
LIFO	HL	Last-in-first-out. See also the definitions of "stack", "queue" and "FIFO".
linked list	HL	A data structure technique of storing data in different areas of memory rather than in a contiguous block and keeping track of the data using pointers.

linker	HL	A utility program that brings together the object modules, operating system routines and other utility software to produce a complete, executable program.
loader	HL	A program that copies an object program held in memory into the memory area designated by the operating system for execution.
local area network (LAN)		A computer network where all the computers are directly linked by cables and/or microwave transmission. This is usually located on a user's premises within a limited geographical area. See also the definition for "wide area network (WAN)".
local variable		A variable that is defined and is capable of being used only in one specified program block.
logic circuit	HL	A circuit whose output can be determined by knowing the input and by following the path through the logic gates.
logic error		An error arising from an incorrect appreciation of the problem leading to an incorrect action being performed and hence a false result being produced.
logic gate	HL	A combinational circuit that performs an elementary logic operation and usually involves one output.
magnetic ink character recognition (MICR)		The identification of characters through the use of magnetic ink. See also the definition of "OCR".
mainframe		A computer, usually in a computer centre, with extensive capabilities and resources to which other computers may be connected so that they can share facilities.
master file		A permanent file holding information that can be accessed and that is periodically updated by processing with a transaction file. See also the definition for "transaction file".
memory address register (MAR)		Holds the address in memory of the instruction at present being executed.
memory manager	HL	A program that is usually part of the operating system that controls the allocation of memory to various applications. It is particularly important in multi-tasking systems where applications might otherwise cause conflicts, and for implementing virtual machines and virtual memory.

memory mapped I/O	See the definition for “DMA”.
menu	A display of a list of optional facilities that can be chosen by the user in order to carry out different functions in a system.
method	1. The behaviour or operation of an object. 2. The procedure used by an object as specified within the object class. See also the definition for “method signature”.
method signature	The number and types of arguments of a method.
MICR	See the definition for “magnetic ink character recognition”.
microprocessor	An integrated circuit incorporating the main components of a central processor. These circuits are used for microcomputers and small devices controlled by computer.
microwave transmission	A method of electronic communication that does not require cables.
modem	An abbreviation for “modulator/demodulator”: a piece of electronic equipment that converts digital signals from a computer into audio signals that are transmitted over telephone lines, and converts them back again.
modular language	A language in which a complete program can be broken down into separate components (modules), each of which is to some extent self-contained. For example, the scope of variables can be limited to a module and does not extend through the entire program. See also the definition for “top-down design”.
modularity	One aspect of structured programming in which individual tasks are programmed as distinct sections or modules. One advantage is the ease with which individual sections can be modified without reference to other sections.
module	A self-contained subset of a program.
modulo arithmetic	Arithmetic that uses the integer result and integer remainder of division as two separate entities.
multi-tasking	A mode of operation that provides for concurrent performance, or interleaved execution, of two or more tasks.
multi-user system	A system that allows two or more people to use the services of a processor within a given period of time.

multi-processing		The simultaneous execution of two or more computer programs or sequences of instructions by a computer (parallel processing).
nand	HL	The output of "nand" is False only if all inputs are True, otherwise the output is True.
network		Any set of interconnected computer systems that share resources and data. See also the definitions for "networking", "local area network (LAN)" and "wide area network (WAN)".
networking		Making use of the services of a network. See also the definitions for "network", "local area network (LAN)" and "wide area network WAN").
node		1. In the terminology of tree structures, each position in the tree is called a "node". 2. Any device on a computer network that can be addressed so that it can be contacted by other computers. 3. A "host" computer on a network.
nor	HL	The output of "nor" is True if all statements are False, False if at least one statement is True.
not		The output of "not" for a statement P is True if P is False, False if P is True.
object		An object is a combination of data and the operations that can be performed in association with that data. Each data part of an object is referred to as a data member while the operations can be referred to as methods. The current state of an object is stored in its data members and that state should only be changed or accessed through the methods. Common categories of operations include: the construction of objects; operations that either set (mutator methods) or return (accessor methods) the data members; operations unique to the data type; and operations used internally by the object.
object-oriented programming (OOP)		An approach to programming in which units of data are viewed as active "objects" rather than the passive units envisioned by the procedural paradigm.
OCR Optical character recognition (reader)		Refers to the use of devices and software to "read" characters and translate them into ASCII characters for later processing. Applications of OCR include the scanning of printed documents to convert the text into digital ASCII text that can then be edited in word processors.

OMR forms		Optical mark and read forms.
on-line		When a user has access to a computer via a terminal.
on-line processing (interactive)		Data processing in which all operations are performed by equipment directly under the control of a central processor, for example, airline reservations.
open systems interconnection (OSI)	HL	A set of protocols allowing different types of computers to be linked together.
operand	HL	In an arithmetical expression, the operand is the data that is to be operated on.
operating system (OS)		Software that controls the execution of programs and that may provide services such as resource allocation, scheduling, input/output control, and data management.
operator	HL	A character or string of characters that designate an operation. See also the definitions for “binary operator” and “unary operator”.
operator precedence		In programming languages, an order relation defining the sequence of the application of operators within an expression.
or		The output of “or” is True if at least one input is True, otherwise the output is False.
overflow	HL	The generation of a quantity, as a result of an arithmetic operation, that is too large to be contained in the result location. See also the definition for “underflow”.
packet	HL	A group of bits made up of control signals, error control bits, coded information and the destination for the data.
packet switching	HL	A method of transmitting data in which the data packet is transmitted as one entity irrespective of the whole message.
parallel interface	HL	An interface through which a computer transmits or receives data that consists of several bits sent simultaneously on separate wires. See also the definition for “serial interface”.
parameter	HL	A parameter is passed to a routine or method by variable name and type. When the code is run, the parameter is replaced by the value of the variable, and becomes the argument of the routine, referred to by the variable name in the definition.

parameter passing		The assignment of values to parameters to be used in a procedure.
parent (node)	HL	The node immediately above a given node, at the next level up. There can only be one parent node for each node, but different nodes may share the same parent.
parity bit		A binary digit appended to a group of binary digits to make the sum of all the digits, including the appended binary digit, either odd or even as established beforehand.
parsing	HL	The breaking down of high-level programming language statements into their component parts during the translation process. An example would be identifying reserved words and variables.
partially-indexed file	HL	A file in which records are ordered in groups. Sequential access to an index followed by direct access to the first record in the group, then sequential access to the desired record, retrieves a particular record. See also the definition for "fully-indexed file".
pass-by-reference		The parameter-passing mechanism by which the address of a variable is passed to the subprogram called. If the subprogram modifies the formal parameter, the corresponding actual parameter is also changed. In Java, all objects, including arrays, are passed-by-reference. See also the definition for "pass-by-value".
pass-by-value		The parameter-passing mechanism by which a copy of the value of the actual parameter is passed to the called procedure. If the called procedure modifies the formal parameter, the corresponding actual parameter is not affected. In Java, all primitives are passed-by-value. See also the definition for "pass-by-reference".
peripheral device		Any device that can communicate with a particular computer, for example: input/output units, auxiliary storage, printers.
pointer	HL	A reference to an address that enables the retrieval of a data item or record. Used in dynamic data structures to move from item to item.
pointing device		An instrument, such as a mouse, trackball or joystick, used to move an icon (sometimes in the form of an arrow) on the screen.
polling	HL	Interrogation of devices for such purposes as avoiding contention, determining operational status, or determining readiness to send or receive data.
polymorphism	HL	The ability of different objects to respond appropriately to the same operation.

pop	HL	To remove an item from the top of a stack.
port	HL	An access point for data entry or exit.
postfix notation	HL	A method of forming mathematical expressions in which each operator is preceded by its operands and indicates the operation to be performed on the operands or the intermediate results that precede it; for example, A added to B and the sum multiplied by C is represented by the expression AB+C*. See also the definitions for “infix notation” and “prefix notation”.
post-order traversal	HL	Traversal of a tree by visiting the nodes recursively in the order left-child, right-child, parent. See also the definitions for “pre-order traversal” and “in-order traversal”.
prefix notation	HL	A method of forming mathematical expressions in which each operator precedes its operands and indicates the operation to be performed on the operands or the intermediate results that follow it. See also the definitions for “infix notation” and “postfix notation”.
pre-order traversal	HL	Traversal of a tree by visiting the nodes recursively in the order parent, left-child, right-child. See also the definitions for “in-order traversal” and “post-order traversal”.
primary memory		The part of the memory where the data and programs that are in use at the time are stored.
primitive data type		Integer, real, character or Boolean data types.
private class members		Members of a class that are only accessible from methods inside the class.
program counter	HL	A register that holds the address of the next instruction to be fetched in the fetch execute cycle.
protocol		An internationally agreed set of rules to ensure transfer of data between devices. A standard protocol is one that is recognized as the standard for a specific type of transfer. For example, TCP/IP.
prototyping		The construction of a simple version of a system in the design stage, showing the user interface but without full processing behind it. This allows the user to propose changes at the design stage.
pseudocode		An artificial language used to describe computer program algorithms without using the syntax of any particular language. During the development of an algorithm, pseudocode often contains sections in natural language that will be replaced later.

public class members		Members of a class that are accessible from anywhere and from any class.
push	HL	To add an item to the top of a stack.
queue	HL	An abstract data structure where items are inserted at one end and retrieved from the other end (FIFO). (The standard operations are given in 5.2.7.)
quicksort	HL	A sort in which a list is first partitioned into lower and upper sublists for which all keys are, respectively, less than some pivot key or greater than the pivot key. See also the definitions for “bubble sort”, “selection sort” and “insertion sort”.
real-time processing		The manipulation of data that is required or generated by some process while the process is in operation; usually the results are used to influence the process, and perhaps related processes, while it is occurring.
record		An aggregate that consists of data objects, possibly with different attributes, that usually have identifiers attached to them. See also the definition for “field”.
recursion	HL	The process whereby a method refers to itself. In many programming languages, a procedure or function can call itself.
reference	HL	Contains the location in memory of an object. The object can contain many individual data members.
register	HL	A part of internal storage that has a specified storage capacity and is usually intended for a specific purpose.
requirements specification		A document that sets out the customer requirements of a computer system. It is written as part of the systems analysis and can be used later to evaluate the system when implemented.
right-child	HL	In a tree, the node to the immediate right of a parent node. See also the definitions for “parent” and “left-child”.
robotics		The techniques used in designing, building and using robots.
robustness		A term used to describe the ability of a program to resist crashing due

router		A device that identifies the destination of messages and sends them via an appropriate route.
search engine		A program that searches a large database to find matching items. The most common use of a search engine is to find Internet addresses based on given key words.
secondary memory		A type of memory that allows a user to store data and programs for as long as desired, in, for example, a hard disk drive.
sector	HL	The smallest accessible storage unit on a disk. The point at which the sector intersects with a track is used to reference the location.
security		Security in the context of computing is a large subject but in outline it might refer to: 1. risk to hardware 2. risk to software 3. risk to information.
seek time	HL	In a disk drive, the time taken for the read/write heads to position themselves over the appropriate track. See also the definition for "rotational delay".
selection sort		A sort in which the items in a set are examined to find an item that fits specified criteria. This item is appended to the sorted set and removed from further consideration, and the process is repeated until all items are in the sorted set. See also the definitions for "bubble sort", "insertion sort" and "quicksort".
semantics		The relationships of characters or groups of characters to their meanings, independent of the manner of their interpretation and use.
sensor		A device that detects measurable elements of a physical process for transfer to a computer.
sentinel	HL	A special value that marks the end of a set of data. Also called an "end of data marker" or "rogue value".
sequential access		An access method in which records are read from, written to, or removed from a file based on the logical order of the records in the file.
sequential file		A file in which records are ordered and are retrieved using sequential access.

sequential search		A search in which records in a file or in another data structure are examined one by one in the order in which they were entered until a specified criterion is met or until there are no more records to examine. See also the definition for "binary search".
serial interface	HL	An interface through which a computer transmits or receives data, one bit at a time. See also the definition for "parallel interface".
server		1. A program that provides services requested by client programs. 2. A computer that provides services to another computer connected over a network.
signature		A combination of specifiers, the method name and the parameter list, that uniquely identifies the method.
simulation		The use of a data processing system to represent selected behavioural characteristics of a physical or abstract system.
single-tasking		A mode of operation that allows only one program to be in use at any time.
single-user system		A system that only allows one user at a time.
software design		The systematic application of scientific and technological knowledge, methods and experience to the design, implementation, and testing of software to optimize its production and support.
software reuse	HL	Creating classes that operate on a wide variety of different objects, and can be "dropped into" a current project, leading to reduced software cost and increased reliability.
speech recognition (voice recognition)		A process of comparing spoken words with those stored in the system.
stack	HL	An abstract data structure where only the top is accessible for the insertion and retrieval of items (LIFO).
star topology		A network in which each device is connected to a central hub. See also the definitions for "tree topology" and "bus topology".
static data structure		Data structures of which the size and nature are determined before a program is executed.
storage requirements		A description of how much memory is required during the running of the program.

storyboard		A diagrammatic form of a prototype showing a planned sequence of screen displays, demonstrating the different paths available to the user.
structure diagram		A diagram that represents the working relationships between the parts of a system or program.
subclass	HL	A class that extends the attributes and methods of a parent class.
subprogram		A program invoked by another program.
subtree	HL	A tree that is part of another tree.
superclass	HL	A class that provides its attributes and methods to a subclass.
syntax		The rules that govern the structure of language statements; in particular, the rules for forming statements in a source language correctly.
syntax error		An error in the rules that govern the structure of language statements.
system documentation		Documentation of the result of the systems analysis stage giving the purpose of the system, the required inputs and outputs, a test plan and the results that are expected.
system life cycle		The course of development changes through which a system passes from its conception to the termination of its use; for example, the phases and activities associated with the analysis, acquisition, design, development, testing, integration, operation, maintenance, and modification of a system.
systems analyst		A person who carries out a systematic investigation of a real or planned system to determine the information requirements and processes of the system, and how these relate to each other and to another system.
systems design		The investigation and recording of existing systems and the design of new systems.
systems flowchart		A flowchart used to describe a complete data processing system, with the flow of data through the clerical operations involved, down to the level of individual programs, but excluding details of such programs.
TCP/IP (transmission control protocol/ Internet protocol)		A set of communications protocols used to connect hosts on the Internet.

top-down design		A method of solving a problem by breaking it down into smaller subproblems. These are then broken down in turn until ultimately a pseudocode representation is obtained that can be used as a basis for program construction. See also the definition for "modular language".
trace		A record of the execution of a computer algorithm exhibiting the sequences in which the instructions were executed.
track	HL	A series of concentric rings placed on a disk surface by the operating system.
transaction file		A temporary file holding data that is later used for processing, generally to update a master file. See also the definition for "master file".
translator		A computer program that transforms all or part of a program expressed in one programming language into another programming language or into a machine language suitable for execution. See also the definitions for "compiler" and "interpreter".
tree	HL	A non-linear data structure (representing a strictly hierarchical system of data) where each data item is thought of as a node.
tree topology		A network that combines the characteristics of bus and star topologies. Groups of star topologies are connected to a central cable. See also the definitions for "star topology" and "bus topology".
truncation	HL	1. The process of approximating a number by ignoring all information beyond a set number of significant figures. Truncation error is the error introduced by this process. 2. The deletion or omission of a leading or a trailing portion of a string in accordance with specified criteria.
truth table	HL	A table that describes a logic function by listing all possible combinations of input values and indicating the output value for each combination.
two's complement	HL	A method of representing negative numbers in the binary system.
unary operator	HL	An operator requiring only one operand to give a single result; for example, negation (overbar for a Boolean expression). See also the definition for "binary operator".
unbalanced tree	HL	A tree in which the right and left subtrees have heights differing by more than one. See also the definition for "balanced tree".

underflow	HL	The generation of a result whose value is too small for the range of the number representation being used. See also the definition for "overflow".
Unicode		A standardized 16-bit character set that represents the character sets of most major languages in the world. See also the definition for "ASCII".
user-defined methods		Methods written by the user which are not inherent to the language.
user-defined objects		Objects whose members and methods are defined by the user and not inherent in the language.
user interface		Hardware, software, or both, that allow a user to interact with and perform operations on a system, program, or device.
utility		A program designed to perform an everyday task such as copying data from one storage device to another.
validation (data input)		The process of checking, with software, that the data input is of the right type and within reasonable limits. See also the definition for "verification (data input)".
variable-length records	HL	Records whose length is not determined in advance. Each record is allocated the space that it needs to store the information it holds. See also the definition for "fixed-length record".
verification (data input)		A method of ensuring that the data in the computer system is the same as the original source data. This may be done by double entry. See also the definition for "validation (data input)".
virtual memory		The use of secondary memory as if it were primary memory.
virus		A program that infects other programs or files by embedding a copy of itself into the target files.
virus checker		A utility program that seeks out and eliminates known viruses.
wide area network (WAN)		A network that provides communication services to a geographic area larger than that served by a local area network or a metropolitan area network, and that may use or provide public communication facilities. See also the definition for "local area network (LAN)".
word		A group of bits that can be addressed, transferred and manipulated as a single unit by the central processing unit.

xor	HL	(Exclusive **or** gate.) The output is True if the two inputs are different; the output is False if the two inputs are alike.

Adapted and reprinted by permission of Pearson Education Limited.

APPENDIX 2

Java examination tool subset (JETS)

The computer science syllabus requires students to learn Java. This does **not** mean **all** of Java. With the many libraries and classes, and the constant changing of the language, that would be impractical. The intention is not for the students to become Java "experts". Rather Java provides a platform for students to develop and demonstrate their understanding of fundamental **algorithm** concepts. Therefore students must only learn a small subset of the entire language, referred to as JETS.

Teachers should note that sample algorithms can be found in the teacher support material for this course.

Only the commands, symbols, and constructs specified in JETS will appear in examination questions. Students will not be required to read or write answers involving other classes or methods. As the program dossier is also written in Java, students will have learned some other constructs and classes, and may choose to use them in their examination answers. However, some classes and methods are specifically forbidden, as they contain commands that would perform tasks that DP students are expected to program from simpler constructs. Students must not use other powerful constructs to avoid coding algorithms. For example, the **java.util** sorting methods are not to be used to answer a question requiring students to create a sorting algorithm.

JETS also specifies a **naming convention** and **style** for examination questions. Teachers should familiarize their students with JETS, including the naming and style conventions. These conventions are intended to make examination questions clear and easily readable. Students are not required to follow these conventions in their answers. However, they should write answers in a clear, consistent and readable style, and must not use non-st

The presentation of JETS

Style Conventions

The style conventions to be used in all examination papers will be as follows:

- examination questions and general rubric will be printed in Times New Roman (proportional) font (12 point). Some general examination rubric will be printed in *italics*. `JETS-code will be printed in Courier (fixed spacing) font 10.5 point`
- all reserved words will be written in **`lower case bold`**
- Class names will always start with a Capital letter
- variable and method names will always start with a small letter
- `multiWordIdentifiers` will use embedded capitals to separate the words (not underscores)
- identifiers will generally use whole words, not abbreviations or acronyms
- proper indentation will always be used
- the order of modules is irrelevant, but the **main** and/or **constructor** method will always be placed at the top
- some examination questions may include statements such as, "recall that ...". The intention is to remind students of any uncommon commands, for example: ***Recall that*** **`String.indexOf(String)`** ***can be used to find the position of one string inside another, like this***:

```
String email = "exams@ibo.org";
int atSign = email.indexOf("@");  //result is 5
```

- non-standard language elements (library classes) may be explained by writing: "A library provides the method|data-type ...", followed by an explanation and example.

For French and Spanish versions of examination papers

- reserved words will remain in English
- string constants will be translated
- user-defined identifiers (class, variable and method names) will be translated as appropriate.

The syntax of JETS

Operators

Arithmetic: + , - , * , / , % (students must understand the polymorphic behaviour of the division operator, for example int / int ==> int)

Relational: == , > , < , >= , <= , !=

Boolean : ! , && , ||

(bitwise boolean operators & , | are not required)

Operator precedence

The standards for operator precedence in Java are assumed knowledge. Examination questions may use extra parentheses for clarity and students should be encouraged to do the same in their solutions.

Notation for literals (values)

```
string  : "in quotation marks"
char    : 's' // in single quotes
integer : 123456 or -312
double  : 124.75 (fixed point) or 1.2475E+02 (floating point)
boolean : true , false
```

Constant identifiers will be written in ALL_CAPS, using an underscore to separate words. They will be defined using **`final static`** , as :

```
final static double NATURAL_LOG_BASE = 2.1782818;
```

Primitive data types

`byte int long double char boolean`

(**`short`** and **`float`** are not included)

Structured data types

`String` class

`StringBuffer` class

Linear Arrays : `int[ ] numbers = new int[100];`

(array of 100 integers, index 0..99)

2-D arrays: `int[ ][ ] checkers = new int[8][8];`

Text files (sequential files)

Random Access files (fields as primitive types)

**** The numeric wrapper classes Integer, Double, and so on, will only be used to provide the functionality of static methods for doing type conversions, as demonstrated in the IBIO methods (below).

Parameter passing

Follows the standard specification in Java, for example, primitive types are automatically passed by value, and structured types (arrays and objects) are always passed by reference.

Symbols

```
/* multi-line
comments   */

// single line
// comments
```

( ) round brackets for parameters

[ ] square brackets for subscripts in arrays

. dot notation for dereferencing object methods and data members

{ } for blocks of code

{ 1 , 2 , 3 } for initializing an array

The assumed set of IBIO commands is listed below.

Input Methods

All input methods display a prompt String, accept keyboard input until the user presses the [enter] key, and then return a value of the specified type. It is assumed that the input routines cannot cause a run-time error. If the user types a String that cannot be converted to the correct type, the input routine returns a default value, for example, a blank String, a 0 numeric value, and so on.

```
String inputString(String prompt)
String input(String prompt)
String input()  // does not print a prompt before inputting
char inputChar(String prompt)
boolean inputBoolean(String prompt)
byte inputByte(String prompt)
int inputInt(String prompt)
long inputLong(String prompt)
double inputDouble(String prompt)
output( String )       -->  outputs a String
output( char )         -->  outputs a char value
output( boolean )      -->  outputs a boolean value
output( byte )         -->  outputs a byte value
output( int )          -->  outputs an int value
output( long )         -->  outputs a long value
output( double )       -->  outputs a double value
```

JETS also uses the System console output commands :

```
System.out.print(String )
System.out.println(String )
// System.in.read() is not included in JETS, although it is used inside
// IBIO.
```

Loops and decisions

```
if (boolean condition)
     { ... commands ... }
else if (boolean condition)
     { ... commands ... }
else
     { ... commands ... } ;

//  switch..break.. is not included in JETS, but students may use it
//  in their answers if they wish.

for ( start; limit; increment)
     { ...commands... } ;

while (boolean condition)
     {...commands... } ;

do
     { ...commands... }
while (boolean condition) ;
```

Files

Standard level/Higher level

BufferedReader(FileReader) -will be used to open a sequential file for input

```
.ready
.read
.readLine
.close
```

PrintWriter(FileWriter) -will be used to open a sequential file for output

```
.ready
.print
.println
.close

// Serialization is not required.
```

Higher level only

RandomAccessFile

constructor: `randomAccessFile(String filename, String accessMode)`

```
.seek
.length
.read .... readInt, readDouble, readBytes, readUTF
.write ....  writeInt, writeDouble, writeBytes, writeUTF
.close
```

Standard methods and data members

`Math` class

```
.abs,.pow,.sin,.cos,.round,.floor
```

`String` class

+ for concatenation

```
.equals(String)
.substring(startPos, endPos)
.length()
.indexOf(String)
.compareTo(String)
.toUpperCase()
.toLowerCase()
```

Arrays

```
.length
```

(casts)

```
 (int)   (double)   (byte)   (char)
(numeric + "")   // to convert a numeric value to a String
```

Static methods

Students should be aware that **`static`** methods in some classes can be used without instantiating an object, such as using **`Integer.parseInt( stringVal)`** to convert a string to an integer (without instantiating a **`new Integer`**).

Understanding the **new** construct is required. Students must be aware that **new** causes an object to be instantiated, and that this is somewhat different from declaring a primitive data type. They should thoroughly understand the rules for **scope** and **lifetime** of identifier references, and that instances may be **automatically destroyed** and **garbage collected** when they go out of scope. For example, students must understand that a value stored in a method's local variables will be lost when the method returns, and that this value cannot be retrieved by subsequent calls to the method. **Static** is a required concept, but will not be directly tested in code (it may appear, but the meaning in the code will not be directly examined).

Dynamic memory allocation (HL only)

Students must also understand that an object type can be declared without instantiating, and that this reference (pointer) can be reassigned later to either a new instance or to a different existing instance.

Other syntactical issues

Java permits commands to span multiple lines. This may be done in examination questions, but only when it improves clarity and readability, for example in a long parameter list:

```
  public int sortArray( String[ ] names ,
                        int listSize ,
                        char ascendingOrDescending
                       )
```

Brackets will always be lined up, either horizontally or vertically:

```
public void printNumbers()
{ int x = 0;
  while ( x < 10 )
  { output( x ); }   //brackets for loop body lined up horizontally
}                   // brackets for method body lined up vertically
```

Class scope

public, private

*// **implements** and **abstract** are not included*
*// **interface** is not included*

Overall structure of classes

Students must understand the concept of a **constructor** and a **main method**, and the difference between them. They must also understand the concept of **extends**.

Applets will not be examined in coded algorithms, although some **concepts** of applets (for example, security) may be examined.

Error handling

```
try { ...commands... }
catch (Exception e) { ...handle the error... };
```

// Error handling in examinations will be limited to simply outputting an error message, setting a flag, or returning from the method. Complex handling of specific Exception types will not be expected. Only the generic Exception and IOException errors must be trapped.

```
methodName() throws IOException
```

Students must understand the idea of throwing an exception, rather than trapping it with **try**…**catch**….

Algorithms to exemplify the elements of JETS

The examples of algorithms that follow are an attempt to illustrate most of the language elements of JETS. In the actual examinations, most algorithms will be considerably shorter than these examples. These were compiled using the Sun JDK 1.3 (Java 2). They run as Console (text-mode) applications, using a standard library of console I/O methods (IBIO).

```
=============================================================

//- HELLO - demonstrates simplified input/output methods (IBIO) -
public class Hello
{ public static void main(String[] args)
  { new Hello();}

  public Hello()
  { String name = inputString("What is your name?");
    int age = inputInt("How old are you?");
    output("Hello " + name);
    output("In 2010, you will be " + (age + 7) );
  }

//============================================================
//  Below are the IBIO simple input and output methods
//  These are assumed to be copied into the source code for all
//  algorithms. A note at the end of each algorithm reminds
//  students of this fact. Students are required to
//  understand the USE of these methods, not memorize their code.
//============================================================

  static void output(String info)
  {  System.out.println(info);
  }
  static void output(char info)
  {  System.out.println(info);
  }
  static void output(byte info)
  { System.out.println(info);
  }
  static void output(int info)
  {  System.out.println(info);
  }
  static void output(long info)
  {  System.out.println(info);
  }
  static void output(double info)
  {  System.out.println(info);
  }
  static void output(boolean info)
  { System.out.println(info);
  }
```

```
static String input(String prompt)
{  String inputLine = "";
   System.out.print(prompt);
   try
   {inputLine = (new java.io.BufferedReader(
      new java.io.InputStreamReader(System.in))).readLine();}
   catch (Exception e)
   { String err = e.toString();
     System.out.println(err);
     inputLine = "";
   }
   return inputLine;
}
static String inputString(String prompt)
{ return input(prompt);
}
static String input()
{ return input("");
}
static int inputInt()
    { return inputInt(""); }

static double inputDouble()
    { return inputDouble(""); }

static char inputChar(String prompt)
{  char result=(char)0;
   try{result=input(prompt).charAt(0);}
   catch (Exception e){result = (char)0;}
   return result;
}
static byte inputByte(String prompt)
{  byte result=0;
   try{result=Byte.valueOf(input(prompt).trim()).byteValue();}
   catch (Exception e){result = 0;}
   return result;
}
static int inputInt(String prompt)
{  int result=0;
   try{result=Integer.valueOf(
                     input(prompt).trim()).intValue();}
   catch (Exception e){result = 0;}
   return result;
}
static long inputLong(String prompt)
{  long result=0;
   try{result=Long.valueOf(input(prompt).trim()).longValue();}
   catch (Exception e){result = 0;}
   return result;
}
static double inputDouble(String prompt)
{  double result=0;
   try{result=Double.valueOf(
                       input(prompt).trim()).doubleValue();}
   catch (Exception e){result = 0;}
   return result;
}
```

```
  static boolean inputBoolean(String prompt)
  {  boolean result=false;
     try{result=Boolean.valueOf(
                         input(prompt).trim()).booleanValue();}
     catch (Exception e){result = false;}
     return result;
  }
//=========== end IBIO ======================================
}
```

```
//-------------------------------------------------------------
// QUADRATIC finds roots of a quadratic polynomial
//-------------------------------------------------------------

public class Quadratic
{ public static void main(String[] args)
  { new Quadratic();}
  public Quadratic()
  { int a = inputInt("A? ");
      int b = inputInt("B? ");
      int c = inputInt("C? ");
      if (isSolvable(a,b,c))
      { output("x1 = " + bigRoot(a,b,c));
        output("x2 = " + smallRoot(a,b,c));
      }
      else
      { output("No roots");}
      input("--- press [enter] ---");
  }

  boolean isSolvable(int a, int b, int c)
  { if ((a != 0) && (discriminant(a,b,c) < 0))
      { return false; }
      else
      { return true; }
  }

  double discriminant(int a, int b, int c)
  { return b*b - 4*a*c;
  }

  double smallRoot(int a, int b, int c)
  { return (-b - Math.pow(discriminant(a,b,c),0.5) ) / (2*a);
  }

  double bigRoot(int a, int b, int c)
  { return (-b + Math.pow(discriminant(a,b,c),0.5) ) / (2*a);
  }

//-------------------------------------------------------------
//---- IBIO - include simplified input and output methods ----
//-------------------------------------------------------------
}
```

```
//-------------------------------------------------------------
// NameSaver sample algorithm – input a list of names into an array.
// XXX" ends the input, then the list is stored in a sequentialfile.
//    This class does not attempt to handle the possible IOExceptions,
//(e.g. locked file or full disk drive) but simple "throws" them.
//-------------------------------------------------------------

import java.io.*;

public class NameSaver
{ public static void main(String[] args) throws IOException
  { new NameSaver();}

  String names[] = new String[1000];
  int namesCount = 0;

  public NameSaver() throws IOException
  { inputNames();
      saveNames();
  }

  void inputNames()
  { String thisName = "";
      namesCount = 0;
      do
      { output("Type a name");
        thisName = input();
        if (!thisName.equals("XXX"))
        { names[namesCount] = thisName;
            namesCount = namesCount + 1;
        }
      } while (!thisName.equals("XXX") && (namesCount < 1000));
  }

  void saveNames() throws IOException
  { PrintWriter outFile = new PrintWriter(
          new FileWriter("namelist.txt"));
      for (int c = 0; c < namesCount; c++)
      { outFile.println(names[c]);
      }
      outFile.close();
  }

//-------------------------------------------------------------
//---- IBIO - include simplified input and output methods ----
//-------------------------------------------------------------
}
```

```
  //-------------------------------------------------------------
  //  ENCRYPT – Encrypts a string by counting the length, adding that
  // number to the ASCII code of each CAPITAL letter.Then the result is
  // printed backward. Only CAPITAL LETTERS get changed.
  //    "HOT2Day" --> adding 7 --> "OVA2Kay" --> "yaK2AVO"
  //-------------------------------------------------------------
  public class Encrypt
  { public static void main(String[] args)
    { new Encrypt();}

    public Encrypt()
    { String message, coded;
      output("Type a message");
      message = input();
      coded = encrypt(message);
      output(reverse(coded));
      input("---- press [enter] ----");
    }

    String encrypt(String message)    // Strings are immutable, so
    { int p,num;                     // use a StringBuffer to
      char codeChar;                 // allow changing single chars
      StringBuffer text = new StringBuffer(message);

      num = text.length();
      for(p = 0; p < num; p++)
      { codeChar = addCode( text.charAt(p), num );
        text.setCharAt(p,codeChar);
      }
      return text.toString();
    }

    char addCode(char letter,int change)
    { if ((letter >= 'A') && (letter <= 'Z'))   // chars behave like
      { char oldCode = (char)(letter - 'A') ;   // ints, arithmetic
                                                // can be performed
        char newCode = (char)((oldCode + change) % 26);

        return (char)('A' + newCode);    // (char) cast required to
      }                                  //  avoid warning messages
      else
      { return letter; }
    }

    String reverse(String message)
    { String backward = "";
      for(int c = message.length() - 1; c >= 0; c = c-1)
      { backward = backward + message.charAt(c);
      }
      return backward;
   }

//-------------------------------------------------------------
//---- IBIO - include simplified input and output methods ----
//-------------------------------------------------------------
}
```

```
//---------------------------------------------------------------
// FileSorter demonstrates using RandomAccessFile to store "records".
// Java does not contain a specific construct like STRUC or RECORD.
// An "inner class" can be used for this purpose.  There is no
// command available to read or write "records" to random access
// files, so these must be programmed, writing one field at a time.
//---------------------------------------------------------------
import java.io.*;   // contains all file-oriented classes and methods

public class FileSorter
{ public static void main(String[] args) throws IOException
  { new FileSorter();}

  public FileSorter() throws IOException
  {
   RandomAccessFile ranFile = new RandomAccessFile("Items.dat","rw");
   create(ranFile);
    System.out.println("--- Records before sorting ---");
    display(ranFile);
    sort(ranFile);
    System.out.println("--- Records after sorting ---");
    display(ranFile);
    ranFile.close();
  }

  class Item   //----- inner class simulates "records" -----------
  { int id;                  // Item class contains 3 data fields
    String name;             //  which will be written into and
    double price;            //  read from the random access file

    final static int NAMELENGTH = 20;
    final static int RECORDSIZE = NAMELENGTH*2 + 12;
                         // constants used to calculate SEEK values

    void readFromFile(RandomAccessFile ranFile, long recordNum)
    //-------------------------------------------------------------
    // Reads one record from ranFile, which must already be open
    // Reads each field - id, price, name – use TRIM to remove
    // padding spaces. IOExceptions are detected and reported
    //-------------------------------------------------------------
    { try
      { ranFile.seek( recordNum * RECORDSIZE);
        id = ranFile.readInt();
        price = ranFile.readDouble();
        StringBuffer nameBuffer = new StringBuffer(Item.NAMELENGTH);
        nameBuffer.setLength(NAMELENGTH);
        for (int c = 0; c < NAMELENGTH; c++)
        { nameBuffer.setCharAt(c, ranFile.readChar());
        }
        name = nameBuffer.toString().trim();
      }
      catch(IOException exc)
      { System.out.println("While reading record # " + recordNum);
        System.out.println(exc.toString());
      }
    }
```

```
  void writeToFile(RandomAccessFile ranFile, long recordNum)
  //-------------------------------------------------------------
  // Writes one record into ranFile, which must already be open
  // IOExceptions are detected and reported
  //-------------------------------------------------------------
  { try
    { ranFile.seek( recordNum * RECORDSIZE);
      ranFile.writeInt(id);
      ranFile.writeDouble(price);
      ranFile.writeChars(setLength(name,NAMELENGTH));
    }
    catch(IOException exc)
    { System.out.println("While writing " + exc.toString()); }
  }

  String setLength(String s,int len)
  //-------------------------------------------------------------
  // Forces length of string to a specific value
  // Necessary before writing into a random-access file
  //-------------------------------------------------------------
  { StringBuffer sb = new StringBuffer(s);
    sb.setLength(len);
    return sb.toString();
  }
}  //---- end of Item class -----------------

void create(RandomAccessFile ranFile) throws IOException
//-------------------------------------------------------------------
// Puts records into ranFile, which must already be open
//-------------------------------------------------------------------
{ Item thisRec = new Item();
  for (int c=0; c < 2; c++)
  { thisRec.id = inputInt();
    thisRec.name = input();
    thisRec.price = inputDouble();
    thisRec.writeToFile(ranFile,c);
  }
}

void display(RandomAccessFile ranFile)
//-------------------------------------------------------------------
// Reads all records from ranFile and prints the fields
//-------------------------------------------------------------------
{ try
  { long recordCount = ranFile.length() / Item.RECORDSIZE;
    Item thisRec = new Item();
    for (int c=0; c < recordCount; c++)
    { thisRec.readFromFile(ranFile, c);
      System.out.println(thisRec.id + ":" + thisRec.name
                         + "=" + thisRec.price);
    }
  }
  catch (IOException exc)
  { System.out.println(exc.toString());}
}
```

```
    void sort(RandomAccessFile ranFile)
    //-----------------------------------------------------------------
    // Bubble sort ranFile, sorting name fields in ascending order
    //-----------------------------------------------------------------
    { try
      { long recordCount = ranFile.length() / Item.RECORDSIZE;
        Item thisRec = new Item();
        Item nextRec = new Item();
        for (int pass = 0; pass < recordCount; pass++)
        { for (int pos = 0; pos < recordCount-1; pos++)
          { thisRec.readFromFile(ranFile,pos);
            nextRec.readFromFile(ranFile,pos+1);
            if (thisRec.name.compareTo(nextRec.name)>0)
            { nextRec.writeToFile(ranFile,pos);
              thisRec.writeToFile(ranFile,pos+1);
            }
          }
        }
      }
      catch (IOException exc)
      { System.out.println(exc.toString());}
    }

//-----------------------------------------------------------
//---- IBIO - include simplified input and output methods ----
//-----------------------------------------------------------
}
```

```
//---------------------------------------------------------------------
//  FactorTree sample algorithm – generates a prime-factor tree
//  This algorithm is for HL candidates only, as binary trees
//  do not appear in the SL syllabus.
//---------------------------------------------------------------------
public class FactorTree
{ public static void main(String[] args)
  { new FactorTree();}

  class Node                        // Use an "inner class" as a
  { int data;                       // Data-Structure similar to a
    Node leftChild;                 // RECORD or STRUC in
    Node rightChild;                // traditional HL languages
  }

  public FactorTree()
  { int number;
    Node root = null;
    number = inputInt("Type an integer:");
    if (number > 2)
     { root = makeTree(number);
       output("The prime factors are");
       showFactors(root);
     }
    output("-----------------");
    outline(root,"");
    input("");
  }

  Node makeTree(int number)    // Recursively create factor tree
  { Node temp = new Node();    // creates a Node (allocates memory)
    temp.leftChild = null;
    temp.rightChild = null;
    temp.data = number;
    int count = 1;
    int fac = 0;
    while (count*count <= number)
    { if ( (number % count) == 0 )
       { fac = count; }
      count = count + 1;
    }
    if (fac > 1)
    { temp.leftChild = makeTree(fac);
      temp.rightChild = makeTree(number / fac);
    }
    return temp;
  }
```

```
  void showFactors(Node here)
  { if (here == null) { output("null"); return;}
    if ( (here.leftChild == null) && (here.rightChild == null))
    { output(here.data);
    }
    else
    { showFactors(here.leftChild);
      showFactors(here.rightChild);
    }
  }


void outline(Node here,String indent)   // Pre-order traversal prints
  { output(indent + here.data);        // tree in "outline" format
    if (here.leftChild != null)
     {outline(here.leftChild, indent + "   ");}
    if (here.rightChild != null)
     {outline(here.rightChild, indent + "   ");}
    }

//-------------------------------------------------------------
//---- IBIO - include simplified input and output methods ----
//-------------------------------------------------------------
}
```

```
//-----------------------------------------------------------------
// This Calendar class is used by a company for scheduling meetings,
// deadlines, deliveries, etc.  All functions will accept dates in
// a variety of formats ("December 25, 2002" or "25 Dec 02" or
//                        "12/25/2002")
// but results are always returned in the format "dd MMM yyyy EEE",
// e.g.  "01 Jul 1998 Wed".  This is also accepted for parameters.
//-----------------------------------------------------------------

import java.util.*;
import java.text.*;

public class Calendar
{ private static final long ONE_DAY = (long)24*60*60*1000;

  private static final SimpleDateFormat dateFormatter =
                         new SimpleDateFormat("dd MMM yyyy EEE");

  private static final String holidays[] =
    {"01 Jan","01 Apr","01 May","23 Aug","25 Dec","xxxxxx"};

  public static String normalDate(String date)
  //-------------------------------------------------------------
  // Determines the day of the week (Mon, Tue, Wed, ...) and
  // returns DATE in the standard format dd MMM yyyy EEE
  // For example, normalDate("4/1/2003") --> "01 Apr 2003 Tue"
  // Returns an empty string "" if DATE is not valid.
  //-------------------------------------------------------------
  { try{Date df = new Date(date);
        return normalDate(df);}
    catch(Exception e){return "";}
  }

  private static String normalDate(Date df)
  { try{return dateFormatter.format(df);}
    catch(Exception e){return "";}
  }


  public static int isWorkDay(String check)
  //-------------------------------------------------------------
  // Calls NORMALDATE, to produce dd MMM yyyy EEE.  If WWW is
  // "Sat" or "Sun", the function returns 0 (false).
  // Otherwise, it consults a calendar file to check for
  // holidays, returning 1 for a workday, 0 for a holiday or
  // weekend, and error code -1 if CHECK is not a valid Date.
  //-------------------------------------------------------------
  { String d;
    try { d = normalDate(check); }
    catch (Exception e) { return -1; }

    String target = d.substring(0,6);
    String weekday = d.substring(12,15);
    int workday = 1;
    if (weekday.equals("Sat") || weekday.equals("Sun"))
    { workday = 0; }
    else
    { int c = 0;
```

```
      while (c<5)
      { if (target.equals(holidays[c]))
        { workday = 0; }
        c = c+1;
      }
    }
    return workday;
  }

  public static String nextDay(String date)
  //-------------------------------------------------------------
  // Accepts DATE in various formats, returns the next date in
  // standard format dd MMM yyyy EEE. Returns an empty string
  //  if DATE is not valid (e.g. 1998.37.58)
  // This correctly accounts for end of the month, end of year,
  // leap years, etc. For example:
  //      NEXTDAY("28 Feb 1998 Sat") ----> "01 Mar 1998 Sun"
  // Returns an empty string if DATE is not valid.
  //-------------------------------------------------------------
  { return normalDate(new Date(new Date(date).getTime() + ONE_DAY));
  }

  public static int daysBetween(String first,String second)
  //-------------------------------------------------------------
  // Counts the number of days between two dates, including the
  // ends.  If FIRST is after SECOND, returns a negative number.
  // IF FIRST and SECOND are the same date, returns 1.
  // If FIRST or SECOND are not valid, returns error code 0
  //-------------------------------------------------------------
  { try
    {Date d1 = new Date(first);
     Date d2 = new Date(second);
     return (int)( (long)(d2.getTime() - d1.getTime()) / ONE_DAY);
    }
    catch(Exception exc)
    { return 0; }
  }


  public static String today()
  //-------------------------------------------------------------
  // Returns today's date in standard format dd MMM yyyy EEE
  //-------------------------------------------------------------
  { try
    { Date now = new Date();
      return normalDate(
             new Date(now.getYear(),now.getMonth(),now.getDate()));
    }
    catch (Exception exc)
    { return ""; }
  }
}
```

```
//-----------------------------------------------------------------
// By providing PUBLIC STATIC Methods, another class can use
// these methods without instantiating an object, providing similar
// functionality to traditional library procedures.  Reusability
// and reliability are improved by careful exception handling.
// In an exam question, only the method headers and comments need be
// provided – candidates do not need to know HOW the methods work.
//-----------------------------------------------------------------
```

```
//---------------------------------------------------------------
// WORKDAYS sample algorithm – inputs two dates, counts the number
//   of workdays between the two dates, including the ends.
//   It uses the Calendar class.
//---------------------------------------------------------------

// uses Calendar class, see previous pages
public class WorkDays
{ public static void main(String[] args)
  { new WorkDays();}

  public WorkDays()
  { String first,last,temp,savedFirst;
    int between;
    output( "This algorithm counts the workdays between two dates.");

    first = "";
    while (first.equals(""))        // bad date returns empty string
    { output("Type in the first date:");   // Loop until good date
      first = input();
      first = Calendar.normalDate(first);
    }

    last = "";
    while (last.equals(""))         // bad date returns empty string
    { output("Type in the last date");     // loops until good date
      last = input();
      last = Calendar.normalDate(last);
    }
    between = Calendar.daysBetween(first,last);
    if (between < 0)
    { temp = first;         // Swap FIRST and LAST
      first = last;
      last = temp;
    }
    savedFirst = first;     // save for output message
    between = Calendar.isWorkDay(first);
    output(first);
    while (!first.equals(last))  // Don't compare Strings with ==
    { first = Calendar.nextDay(first);
      between = between + Calendar.isWorkDay(first);
      output(first);
    }
    output(between + " workdays between " + savedFirst + " and " + last
);
  }

//-------------------------------------------------------------
//---- IBIO - include simplified input and output methods ----
//-------------------------------------------------------------
}
```

```
//----- Sample Output -----------------------------------
/*  This algorithm counts the workdays between two dates.
  Type in the first date:
  12/21/2002
  Type in the last date
  12/31/2002
  21 Dec 2002 Sat
  22 Dec 2002 Sun
  23 Dec 2002 Mon
  24 Dec 2002 Tue
  25 Dec 2002 Wed
  26 Dec 2002 Thu
  27 Dec 2002 Fri
  28 Dec 2002 Sat
  29 Dec 2002 Sun
  30 Dec 2002 Mon
  31 Dec 2002 Tue
  6 workdays between 21 Dec 2002 Sat and 31 Dec 2002 Tue */
//----- end Sample Output -------------------------------
```

```
public class TestIBIO
{ public static void main(String[] args)
  { new TestIBIO();}

  public TestIBIO()
  { String theString = inputString("String:");
    if (theString.equals("1"))
    { output("Yes"); }
    else
    { output(theString); }

    char theChar = inputChar("char:");
    if (theChar == '2')
    { output("Yes"); }
    else
    { output(theChar); }

    byte theByte = inputByte("byte:");
    if (theByte == 3)
    { output("Yes"); }
    else
    { output(theByte); }

    int theInt = inputInt("int:");
    if (theInt == 4)
    { output("Yes"); }
    else
    { output(theInt); }

    long theLong = inputLong("long:");
    if (theLong == 5)
    { output("Yes"); }
    else
    { output(theLong); }

    double theDouble = inputDouble("double:");
    if (theDouble == 6)
    { output("Yes"); }
    else
    { output(theDouble); }

    boolean theBoolean = inputBoolean("boolean:");
    if (theBoolean == true)
    { output("Yes"); }
    else
    { output(theBoolean); }

    input("-- press [enter] to quit --");
    }
//-------------------------------------------------------------
//---- IBIO - include simplified input and output methods ----
//-------------------------------------------------------------
}
```

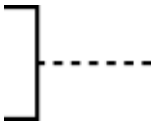
APPENDIX 3

Systems flowchart symbols

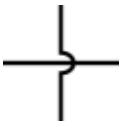
Action or process

Input or output device (description inside)

Annotation



Lines crossing



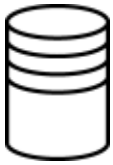
Lines joining

Data flow

Document

Tape

Disk



Online storage

Communication link (two-way unless indicated)

APPENDIX 4

Logic gate symbols