COMPUTER SCIENCE (868)

Aims (Conceptual)

(1) To understand algorithmic problem solving using data abstractions, functional and procedural abstractions, and object based and object-oriented abstractions.

(2) To understand: (a) how computers represent, store and process data at different levels of abstraction that mediate between the machine and the algorithmic problem solving level and (b) how they communicate with the outside world.

(3) To create awareness of ethical issues related to computing and to promote safe, ethical behavior.

(4) To make students aware of future trends in computing.

Aims (Skills)

To devise algorithmic solutions to problems and to be able to code, validate, document, execute and debug the solution using the Java programming system.

CLASS XI

There will be two papers in the subject:

Paper I: *Theory……..* *3 hours…70 marks*

Paper II: *Practical……..* *3 hours…30 marks*

PAPER I –THEORY – 70 MARKS

SECTION A

Basic Computer Hardware and Software

1. Numbers

Representation of numbers in different bases and interconversion between them (e.g. binary, octal, decimal, hexadecimal). Addition and subtraction operations for numbers in different bases.

Introduce the positional system of representing numbers and the concept of a base. Discuss the conversion of representations between different bases using English or pseudo code. These algorithms are also good examples for defining different functions in a class modelling numbers (when programming is discussed). For addition and subtraction (1's complement and 2's complement) use the analogy with decimal numbers, emphasize how carry works (this will be useful later when binary adders are discussed).

2. Encodings

(a) Binary encodings for integers and real numbers using a finite number of bits (sign-magnitude, 2's complement, mantissa-exponent notation).

Signed, unsigned numbers, least and most significant bits. Sign-magnitude representation and its shortcomings (two representations for 0, addition requires extra step); two's-complement representation. Operations (arithmetic, logical, shift), discuss the basic algorithms used for the arithmetic operations. Floating point representation: normalized scientific notation, mantissa-exponent representation, binary point (discuss trade-off between size of mantissa and exponent). Single and double precision.

(b) Characters and their encodings (e.g. ASCII, ISCII, Unicode).

Discuss the limitations of the ASCII code in representing characters of other languages. Discuss the Unicode representation for the local language. Java uses Unicode, so strings in the local language can be used (they can be displayed if fonts are available) – a simple table lookup for local language equivalents for Latin (i.e. English) character strings may be done. More details on Unicode are available at www.unicode.org.

3. Propositional logic, Hardware implementation, Arithmetic operations

(a) Propositional logic, well-formed formulae, truth values and interpretation of well formed formulae, truth tables.

*Propositional variables; the common logical connectives ((not)(negation), ∧ (and)(conjunction), ∨ (or)(disjunction), ⇒ (implication), ⇔ (equivalence)); definition of a well-formed formula (wff); representation of simple word problems as wff (this can be used for motivation); the values **true** and **false**; interpretation of a wff;*

truth tables; satisfiable, unsatisfiable and valid formulae.

(b) Logic and hardware, basic gates (AND, NOT, OR) and their universality, other gates (NAND, NOR, XOR, XNOR), half adder, full adder.

Show how the logic in (a) above can be realized in hardware in the form of gates. These gates can then be combined to implement the basic operations for arithmetic. Tie up with the arithmetic operations on integers discussed earlier in 2 (a).

SECTION B

The programming element in the syllabus is aimed at algorithmic problem solving and **not** merely rote learning of Java syntax. The Java version used should be 5.0 or later. For programming, the students can use any text editor and the javac and java programs or any other development environment: for example, BlueJ, Eclipse, NetBeans etc. BlueJ is strongly recommended for its simplicity, ease of use and because it is very well suited for an 'objects first' approach.

4. Introduction to Object Oriented Programming using Java

Note that topics 5 to 12 should be introduced almost simultaneously along with Classes and their definitions.

5. Objects

(a) Objects as data (attributes) + behaviour (methods or methods); object as an instance of a class.

Difference between object and class should be made very clear. BlueJ (www.bluej.org) and Greenfoot (www.greenfoot.org) can be used for this purpose.

(b) Analysis of some real-world programming examples in terms of objects and classes.

Use simple examples like a calculator, date, number etc. to illustrate how they can be treated as objects that behave in certain well-defined ways and how the interface provides a way to access behaviour. Illustrate behaviour changes by adding new methods, deleting old methods or modifying existing methods.

(c) Basic concept of a virtual machine; Java Virtual Machine (JVM); compilation and execution of Java programs (the javac and java programs).

The JVM is a machine but built as a program and not through hardware. Therefore it is called a virtual machine. To run, JVM machine language programs require an interpreter. The advantage is that such JVM machine language programs (.class files) are portable and can run on any machine that has the java program.

(d) Compile time and run time errors; basic concept of an exception, the Exception class, try-catch, throw, throws and finally.

Differentiate between compile time and run time errors. Run time errors crash the program. Recovery is possible by the use of exceptions. Explain how an exception object is created and passed up until a matching catch is found. This behaviour is different from the one where a value is returned by a deeply nested method call.

6. Primitive values, Wrapper classes, Types and casting

Primitive values and types: byte, int, short, long, float, double, boolean, char. Corresponding wrapper classes for each primitive type. Class as type of the object. Class as mechanism for user defined types. Changing types through user defined casting and automatic type coercion for some primitive types.

Ideally, everything should be a class; primitive types are defined for efficiency reasons; each primitive type has a corresponding wrapper class. Classes as user defined types. In some cases types are changed by automatic coercion or casting – e.g. mixed type expressions. However, casting in general is not a good idea and should be avoided, if possible.

7. Variables, Expressions

Variables as names for values; named constants (final), expressions (arithmetic and logical) and their evaluation (operators, associativity, precedence). Assignment operation; difference

between left-hand side and right-hand side of assignment.

Variables denote values; variables are already defined as attributes in classes; variables have types that constrain the values it can denote. Difference between variables denoting primitive values and object values – variables denoting objects are references to those objects. The assignment operator = is special. The variable on the LHS of = denotes the memory location while the same variable on the RHS denotes the contents of the location e.g. i=i+2.

NOTE: Library functions for solving expressions may be used as and when required.

8. Statements, Scope

Statements; conditional (if, if else, if else if, switch case) ternary operator, looping (for, while, do while), continue, break; grouping statements in blocks, scope and visibility of variables.

Describe the semantics of the conditional and looping statements in detail. Evaluation of the condition in conditional statements.

Nesting of blocks. Variables with block scope, method scope, class scope. Visibility rules when variables with the same name are defined in different scopes.

9. Methods and Constructors

Methods and Constructors (as abstractions for complex user defined operations on objects), methods as mechanisms for side effects; formal arguments and actual arguments in methods; different behaviour of primitive and object arguments. Static methods and variables. The ***this*** operator. Examples of algorithmic problem solving using methods (number problems, finding roots of algebraic equations etc.).

*Methods are like complex operations where the object is implicitly the first argument. Operator **this** denotes the current object. Methods typically return values. Illustrate the difference between primitive values and object values as arguments (changes made inside methods persist after the call for object values). Static definitions as class variables and class methods visible and shared by all instances. Need for static methods and variables. Introduce the main method – needed to begin execution. Constructor as a special kind of method; the new operator; multiple constructors with different argument structures; constructor returns a reference to the object.*

10. Arrays, Strings

Structured data types – arrays (single and multi-dimensional), strings. Example algorithms that use structured data types (searching, finding maximum/minimum, sorting techniques, solving systems of linear equations, substring, concatenation, length, access to char in string, etc.).

Storing many data elements of the same type requires structured data types – like arrays. Access in arrays is constant time and does not depend on the number of elements. Sorting techniques (bubble, selection, insertion), Structured data types can be defined by classes – String. Introduce the Java library String class and the basic operations on strings (accessing individual characters, various substring operations, concatenation, replacement, index of operations).

SECTION C

11. Basic input/output Data File Handling (Binary and Text)

(a) Basic input/output using Scanner and Printer classes.

Input/output exceptions. Tokens in an input stream, concept of whitespace, extracting tokens from an input stream (String Tokenizer class). The Scanner class can be used for input of various types of data (e.g. int, float, char etc.) from the standard input stream. Similarly, the Printer class handles output. Only basic input and output using these classes should be covered.

Discuss the concept of a token (a delimited continuous stream of characters that is meaningful in the application program – e.g. words in a sentence where the delimiter is the blank character). This naturally leads to the idea of delimiters and in particular whitespace and user defined characters as delimiters. As an example show how the StringTokenizer class allows one to extract a sequence of tokens from a string with user defined delimiters.

(b) Data File Handling.

Need for Data file, Input Stream, Output Stream, Byte Stream (FileInputStream and FileOutputStream), Character Stream (FileReader, FileWriter), Operations- Creation, Reading, Writing, Appending, and Searching.

12. Recursion

Concept of recursion, simple recursive methods (e.g. factorial, GCD, binary search, conversion of representations of numbers between different bases).

Many problems can be solved very elegantly by observing that the solution can be composed of solutions to 'smaller' versions of the same problem with the base version having a known simple solution. Recursion can be initially motivated by using recursive equations to define certain methods. These definitions are fairly obvious and are easy to understand. The definitions can be directly converted to a program. Emphasize that any recursion must have a base case. Otherwise, the computation can go into an infinite loop.

13. Implementation of algorithms to solve problems

The students are required to do lab assignments in the computer lab concurrently with the lectures. Programming assignments should be done such that each major topic is covered in at least one assignment. Assignment problems should be designed so that they are sufficiently challenging and make the student do algorithm design, address correctness issues, implement and execute the algorithm in Java and debug where necessary.

Self-explanatory.

14. Packages

Definition, creation of packages, importing user defined packages, interaction of objects across packages.

Java Application Programming Interface (API), development of applications using user defined packages.

15. Trends in computing and ethical issues

(a) Artificial Intelligence, Internet of Things, Virtual Reality and Augmented Reality.

Brief understanding of the above and their impact on Society.

(b) Cyber Security, privacy, netiquette, spam, phishing.

Brief understanding of the above.

(c) Intellectual property, Software copyright and patents and Free Software Foundation.

Intellectual property and corresponding laws and rights, software as intellectual property.

Software copyright and patents and the difference between the two; trademarks; software licensing and piracy. free Software Foundation and its position on software, Open Source Software, various types of licensing (e.g. GPL, BSD).

Social impact and ethical issues should be discussed and debated in class. The important thing is for students to realise that these are complex issues and there are multiple points of view on many of them and there is no single 'correct' or 'right' view.

PAPER II: PRACTICAL – 30 MARKS

This paper of three hours duration will be evaluated internally by the school.

The paper shall consist of three programming problems from which a candidate has to attempt any one. The practical consists of the two parts:

(1) Planning Session

(2) Examination Session

The total time to be spent on the Planning session and the Examination session is three hours. A maximum of 90 minutes is permitted for the Planning session and 90 minutes for the Examination session. **Candidates are to be permitted to proceed to the Examination Session only after the 90 minutes of the Planning Session are over.**

Planning Session

The candidates will be required to prepare an algorithm and a hand-written Java program to solve the problem.

Examination Session

The program handed in at the end of the Planning session shall be returned to the candidates. The candidates will be required to key-in and execute the Java program on seen and unseen inputs individually on the Computer and show execution to the examiner. A printout of the program listing, including output results should be attached to the answer script containing the algorithm and handwritten program. This should be returned to the examiner. The program should be sufficiently documented so that the algorithm, representation and development process is clear from reading the program. Large differences between the planned program and the printout will result in loss of marks.

Teachers should maintain a record of all the assignments done as part of the practical work throughout the year and give it due credit at the time of cumulative evaluation at the end of the year. Students are expected to do a **minimum** of twenty assignments for the year and **ONE** project based on the syllabus.

LIST OF SUGGESTED PROJECTS:

PRESENTATION / MODEL BASED/ APPLICATION BASED

1. Creating an expert system for road-traffic management (routing and re-routing of vehicles depending on congestion).
2. Creating an expert system for medical diagnosis on the basis of symptoms and prescribe a suitable treatment.
3. Creating a security system for age-appropriate access to social media.
4. Simulate Adders using Arduino Controllers and Components.
5. Simulate a converter of Binary to Decimal number systems using Arduino Controllers and Components.
6. Develop a console-based application using Java for Movie Ticket Reservation.
7. Develop a console-based application using Java to encrypt and decrypt a message (using cipher text, Unicode-exchange, etc).
8. Develop a console-based application using Java to find name of the bank and branch location from IFSC.
9. Develop a console-based application using Java to calculate taxable income (only direct tax).
10. Develop a console-based application using Java to develop a simple text editor (text typing, copy, cut, paste, delete).

EVALUATION

Marks (out of a total of 30) should be distributed as given below:

Continuous Evaluation

Candidates will be required to submit a work file containing the practical work related to programming assignments done during the year and **ONE** project.

Programming assignments done throughout the year	10 marks
Project Work (based on any topic from the syllabus)	5 marks

Terminal Evaluation

Solution to programming problem on the computer	15 Marks

(Marks should be given for choice of algorithm and implementation strategy, documentation, correct output on known inputs mentioned in the question paper, correct output for unknown inputs available only to the examiner).

CLASS XII

There will be two papers in the subject:

Paper I: *Theory……..* *3 hours….70 marks*

Paper II: *Practical……..* *3 hours….30 marks*

PAPER I –THEORY – 70 MARKS

SECTION A

1. Boolean Algebra

(a) Propositional logic, well formed formulae, truth values and interpretation of well formed formulae (wff), truth tables, satisfiable, unsatisfiable and valid formulae. Equivalence laws and their use in simplifying wffs.

*Propositional variables; the common logical connectives (~ (not)(negation), ∧ (and)(conjunction), ∨ (or)(disjunction), ⇒ (implication), ⇔ (biconditional); definition of a well-formed formula (wff); `representation of simple word problems as wff (this can be used for motivation); the values **true** and **false**; interpretation of a wff; truth tables; satisfiable, unsatisfiable and valid formulae.*

*Equivalence laws: commutativity of ∧, ∨; associativity of ∧, ∨; distributivity; De Morgan's laws; law of implication ($p \Rightarrow q \equiv \sim p \vee q$); law of biconditional ($(p \Leftrightarrow q) \equiv (p \Rightarrow q) \wedge (q \Rightarrow p)$); identity ($p \equiv p$); law of negation ($\sim(\sim p) \equiv p$); law of excluded middle ($p \vee \sim p \equiv$ **true**); law of contradiction ($p \wedge \sim p \equiv$ **false**); tautology and contingency simplification rules for ∧, ∨. Converse, inverse and contra positive. Chain rule, Modus ponens.*

(b) Binary valued quantities; basic postulates of Boolean algebra; operations AND, OR and NOT; truth tables.

(c) Basic theorems of Boolean algebra (e.g. duality, idempotence, commutativity, associativity, distributivity, operations with 0 and 1, complements, absorption, involution); De Morgan's theorem and its applications; reducing Boolean expressions to sum of products and product of sums forms; Karnaugh maps (up to four variables).

Verify the laws of Boolean algebra using truth tables. Inputs, outputs for circuits like half and full adders, majority circuit etc., SOP and POS representation; Maxterms & Minterms, Canonical and Cardinal representation, reduction using Karnaugh maps and Boolean algebra.

2. Computer Hardware

(a) Elementary logic gates (NOT, AND, OR, NAND, NOR, XOR, XNOR) and their use in circuits.

(b) Applications of Boolean algebra and logic gates to half adders, full adders, encoders, decoders, multiplexers, NAND, NOR as universal gates.

Show the correspondence between Boolean methods and the corresponding switching circuits or gates. Show that NAND and NOR gates are universal by converting some circuits to purely NAND or NOR gates.

SECTION B

The programming element in the syllabus (Sections B and C) is aimed at algorithmic problem solving and **not** merely rote learning of Java syntax. The Java version used should be 5.0 or later. For programming, the students can use any text editor and the javac and java programs or any other development environment: for example, BlueJ, Eclipse, NetBeans etc. BlueJ is strongly recommended for its simplicity, ease of use and because it is very well suited for an 'objects first' approach.

3. Implementation of algorithms to solve problems

The students are required to do lab assignments in the computer lab concurrently with the lectures. Programming assignments should be done such that each major topic is covered in at least one assignment. Assignment problems should be designed so that they are sufficiently challenging. Students must do algorithm design, address correctness issues, implement and execute the algorithm in Java and debug where necessary.

Self explanatory.

4. Programming in Java (Review of Class XI Sections B and C)

Note that items 4 to 13 should be introduced almost simultaneously along with classes and their definitions.

While reviewing, ensure that new higher order problems are solved using these constructs.

5. Objects

(a) Objects as data (attributes) + behaviour (methods); object as an instance of a class. Constructors.

(b) Analysis of some real-world programming examples in terms of objects and classes.

(c) Basic input/output using Scanner and Printer classes from JDK; input/output exceptions. Tokens in an input stream, concept of whitespace, extracting tokens from an input stream (String Tokenizer class).

6. Primitive values, Wrapper classes, Types and casting

Primitive values and types: byte, int, short, long, float, double, boolean, char. Corresponding wrapper classes for each primitive type. Class as type of the object. Class as mechanism for user defined types. Changing types through user defined casting and automatic type coercion for some primitive types.

7. Variables, Expressions

Variables as names for values; named constants (final), expressions (arithmetic and logical) and their evaluation (operators, associativity, precedence). Assignment operation; difference between left hand side and right hand side of assignment.

8. Statements, Scope

Statements; conditional (if, if else, if else if, switch case, ternary operator), looping (for, while, do while, continue, break); grouping statements in blocks, scope and visibility of variables.

9. Methods

Methods (as abstractions for complex user defined operations on objects), formal arguments and actual arguments in methods; different behaviour of primitive and object arguments. Static method and variables. The **this** Operator. Examples of algorithmic problem solving using methods (number problems, finding roots of algebraic equations etc.).

10. Arrays, Strings

Structured data types – arrays (single and multi-dimensional), address calculations, strings. Example algorithms that use structured data types (e.g. searching, finding maximum/minimum, sorting techniques, solving systems of linear equations, substring, concatenation, length, access to char in string, etc.).

Storing many data elements of the same type requires structured data types – like arrays. Access in arrays is constant time and does not depend on the number of elements. Address calculation (row major and column major), Sorting techniques (bubble, selection, insertion). Structured data types can be defined by classes – String. Introduce the Java library String class and the basic operations on strings (accessing individual characters, various substring operations, concatenation, replacement, index of operations). The class StringBuffer should be introduced for those applications that involve heavy manipulation of strings.

11. Recursion

Concept of recursion, simple recursive methods (e.g. factorial, GCD, binary search, conversion of representations of numbers between different bases).

Many problems can be solved very elegantly by observing that the solution can be composed of solutions to 'smaller' versions of the same problem with the base version having a known simple solution. Recursion can be initially motivated by using recursive equations to define certain methods. These definitions are fairly obvious and are easy to understand. The definitions can be directly converted to a program. Emphasize that any recursion must have a base case. Otherwise, the computation can go into an infinite loop.

The tower of Hanoi is a very good example of how recursion gives a very simple and elegant solution where as non-recursive solutions are quite complex.

SECTION C

Inheritance, Interface, Polymorphism, Data structures, Computational complexity

12. Inheritance, Interfaces and Polymorphism

(a) Inheritance; super and derived classes; member access in derived classes; redefinition of variables and methods in subclasses; abstract classes; class Object; protected visibility. Subclass polymorphism and dynamic binding.

Emphasize inheritance as a mechanism to reuse a class by extending it. Inheritance should not normally be used just to reuse some methods defined in a class but only when there is a genuine specialization (or subclass) relationship between objects of the super class and that of the derived class.

(b) Interfaces in Java; implementing interfaces through a class; interfaces for user defined implementation of behaviour.

Motivation for interface: often when creating reusable classes some parts of the exact implementation can only be provided by the final end user. For example, in a class that sorts records of different types the exact comparison operation can only be provided by the end user. Since only he/she knows which field(s) will be used for doing the comparison and whether sorting should be in ascending or descending order be given by the user of the class.

Emphasize the difference between the Java language construct interface and the word interface often used to describe the set of method prototypes of a class.

13. Data structures

(a) Basic data structures (stack, queue, circular queue, dequeue); implementation directly through classes; definition through an interface and multiple implementations by implementing the interface. Conversion of Infix to Prefix and Postfix notations.

Basic algorithms and programs using the above data structures.

Data structures should be defined as abstract data types with a well-defined interface (it is instructive to define them using the Java interface construct).

(b) Single linked list (Algorithm and programming), binary trees, tree traversals (Conceptual).

The following should be covered for each data structure:

Linked List (single): insertion, deletion, reversal, extracting an element or a sublist, checking emptiness.

Binary trees: apart from the definition the following concepts should be covered: root, internal nodes, external nodes (leaves), height (tree, node), depth (tree, node), level, size, degree, siblings, sub tree, completeness, balancing, traversals (pre, post and in-order).

14. Complexity and Big O notation

Concrete computational complexity; concept of input size; estimating complexity in terms of methods; importance of dominant term; constants, best, average and worst case.

Big O notation for computational complexity; analysis of complexity of example algorithms using the big O notation (e.g. Various searching and sorting algorithms, algorithm for solution of linear equations etc.).

PAPER II: PRACTICAL – 30 MARKS

This paper of three hours' duration will be evaluated by the Visiting Examiner appointed locally and approved by CISCE.

The paper shall consist of three programming problems from which a candidate has to attempt any one. The practical consists of the two parts:

1. Planning Session
2. Examination Session

The total time to be spent on the Planning session and the Examination session is three hours. A maximum of 90 minutes is permitted for the Planning session and 90 minutes for the Examination session.

Candidates are to be permitted to proceed to the Examination Session only after the 90 minutes of the Planning Session are over.

Planning Session

The candidates will be required to prepare an algorithm and a hand written Java program to solve the problem.

Examination Session

The program handed in at the end of the Planning session shall be returned to the candidates. The candidates will be required to key-in and execute the Java program on seen and unseen inputs individually on the Computer and show execution to the Visiting Examiner. A printout of the program listing including output results should be attached to the answer script containing the algorithm and handwritten program. This should be returned to the examiner. The program should be sufficiently documented so that the algorithm, representation and development process is clear from reading the program. Large differences between the planned program and the printout will result in loss of marks.

Teachers should maintain a record of all the assignments done as part of the practical work through the year and give it due credit at the time of cumulative evaluation at the end of the year. Students are expected to do a **minimum of twenty-five** assignments for the year.

EVALUATION:

Marks (out of a total of **30**) should be distributed as given below:

Continuous Evaluation

Candidates will be required to submit a work file containing the practical work related to programming assignments done during the year.

Programming assignments done throughout the year (Internal Evaluation)	10 marks
Programming assignments done throughout the year (Visiting Examiner)	5 marks

Terminal Evaluation

Solution to programming problem on the computer	15 Marks

Marks should be given for choice of algorithm and implementation strategy, documentation, correct output on known inputs mentioned in the question paper, correct output for unknown inputs available only to the examiner.

NOTE:

Algorithm should be expressed clearly using any standard scheme such as a pseudo code.

EQUIPMENT

There should be enough computers to provide for a teaching schedule where at least three-fourths of the time available is used for programming.

Schools should have equipment/platforms such that all the software required for practical work runs properly, i.e. it should run at acceptable speeds.

Since hardware and software evolve and change very rapidly, the schools may have to upgrade them as required.

Following are the recommended specifications as of now:

The Facilities:

- A lecture cum demonstration room with a MULTIMEDIA PROJECTOR/ an LCD and O.H.P. attached to the computer.
- A white board with white board markers should be available.
- A fully equipped Computer Laboratory that allows one computer per student.
- Internet connection for accessing the World Wide Web and email facility.
- The computers should have a minimum of 1 GB RAM and a P IV or higher processor. The basic requirement is that it should run the operating system and Java programming system (Java compiler, Java runtime environment, Java development environment) at acceptable speeds.
- Good Quality printers.

Software:

- Any suitable Operating System can be used.
- JDK 6 or later.
- Documentation for the JDK version being used.
- A suitable text editor. A development environment with a debugger is preferred (e.g. BlueJ, Eclipse, NetBeans). BlueJ is recommended for its ease of use and simplicity.

SAMPLE TABLE FOR PRACTICAL WORK

S. No.	Unique Identification Number (Unique ID) of the candidate	Assessment of Practical File		Assessment of the Practical Examination (To be evaluated by the Visiting Examiner only)				TOTAL MARKS (Total Marks are to be added and entered by the Visiting Examiner)
		Internal Evaluation 10 Marks	Visiting Examiner 5 Marks	Algorithm	Java Program with internal Documentation	Hard Copy (printout)	Output	
				3 Marks	7 Marks	2 Marks	3 Marks	30 Marks
1.								
2.								
3.								
4.								
5.								
6.								
7.								
8.								
9.								
10.								

Name of the Visiting Examiner:_____

Signature: _____

Date:_____