# CS2114

Prerequisites and core requisites

**Data types and Operators:** Primitive Datatypes, Declarations, Ranges, Variable Names Conventions, Numeric Literals, Character Literals, String Literals, Arrays(One dimensional; two- dimensional), Array of Object References, Accessing arrays, manipulating arrays, Enumerated Data Types, Non-Primitive Datatypes, Defining a class, variable and method in Java, Method Signature; method calls, Expressions in Java; introduction to various operators, Assignment Operator, Arithmetic Operators, Relational Operators, Logical Operators, Conditional Operators, Operator Precedence, Implicit Type Conversions, Upcasting and downcasting, Strict typing, Type conversion

**Control Flow statements:** if, if-else, if-else-if, switch case, for statement (both flavors traditional and enhanced for), while and do-while loops, continue Statement; labelled continue statement, The break Statement; labelled break statement, return statement

**OOPS and its application in Java:** Classes and Objects, Defining a class, Defining instance variables and methods, Creating objects out of a class, Method calls via object references, Abstraction, Interfaces and Abstract classes, Abstract and non-abstract methods, Inheritance, extends and implements keywords in Java, Super class and Sub class, this keyword, super keyword in Java for inheritance, Concrete classes in Java, Polymorphism, Compile time polymorphism -- Overloading of methods, Run time polymorphism -- Overriding of methods, Method Overriding rules and method overloading rules, Introduction to Object class and it's methods, Encapsulation, Protection of data, Getters/Setters, Memory management in Java (Heap, Stack)

**Packages:** Need for packages, What are packages; package declaration in Java, Import statement in Java, How do packages resolve name clashes?

**Miscellaneous:** Var-Args, Reference variables, local variables, instance variables, Memory allocations to variables, Double equals operator (==) operator for primitives and objects, toString() method on an object

**Statics:** Static variables and methods, Static imports, Static initialization blocks; instance intialization blocks, Static concept in inheritance

**Constructors:** What are Constructors? Properties of Constructors, Default and Parameterized Constructors, Rules for constructor implementation, Constructor Chaining, this call; super call for constructors, Constructors for Enumerated Data Types, Constructors concept for Abstract classes and interfaces

**Exceptions in Java:** What are Exceptions? Need for exceptions, How can Exceptions be coded in Java?

API heirarchy for Exceptions, Types of Exceptions, Keywords in Exception API: try, catch, finally, throw, throws, Rules for coding Exceptions, Declaring Exceptions, Defining and Throwing Exceptions, Errors and Runtime Exceptions

**Strings in Java:** String heap memory and Constant Pool memory, Immutability in Strings, String creation on heap and constant pool, Method APIs on String; operations on Strings, Mutability of String Objects - StringBuilder and StringBuffer, Splitting of Strings and StringTokenizer class

**Collection Framework in Java:** The Collections Framework, The Set Interface, Set Implementation Classes, The List Interface, List Implementation Classes, The Map Interface, Map Implementation Classes, Queue Interface, Queue Implmentation classes, Utility classes, Sorting collections using utility methods, equals() and hashCode contract in Java collections, overriding equals and hashCode methods in Java, Primitive wrapper classes and all its method APIs

**Access Modifiers in Java:** Default, Protected, Private, Public

# Learning Objectives

• Design, implement, test, and debug programs using dynamic data structures such as linked lists, stacks, and queues.

• Design, implement, and test medium-sized programs (e.g., 1K-3K lines of code), including interactive applications.

• Demonstrate time management skills on medium-sized programming projects.

• Use a range of software development tools such as editors, class browsers, bug-tracking debuggers.

• Identify data abstractions appropriate for use in a given problem.

• Develop reusable, type-safe software components.

• Design simple interfaces for data abstractions.

• Design, implement, test, and debug recursive programs for tasks such as implementing binary trees.

• Apply basic algorithm analysis to simple algorithms.

• Recognize and apply simple design patterns.

• Work in a small team to analyze, design, and develop a team project.

# Syllabus

1. Object-oriented software design
   1. • Organizing and managing multi-class designs
   2. • Inheritance, abstract classes, polymorphism
   3. • Interfaces and alternative implementations
   4. • Allocating responsibilities
   5. • Basic design patterns
   6. • Modeling tools
2. Data Structures
   1. • Design, implementation, and use of classic data structures
   2. • Stacks, queues, lists, binary trees, binary search trees
   3. • Comparisons of implementations
3. Algorithms and problem-solving
   1. • Algorithm design techniques
   2. • Linked structures
   3. • Using API classes
4. Introduction to algorithm analysis
   1. • Upper and average complexity bounds
   2. • Big “O” notation
5. Searching and Sorting
   1. • Sequential and binary search
   2. • Simple sorting algorithms
6. Recursion
7. Software engineering issues
   1. • Software project management
   2. • Object-oriented testing
   3. • Pair programming
   4. • Team-based development