Object Oriented Programming with Java

AKTU B.Tech – Exam Oriented Book-style Notes

MyCollegeVerse.in

Building Academic Identity – The Student OS

This book-style document is designed for AKTU B.Tech students preparing for Object Oriented Programming with Java examinations. It focuses on patterns seen in recent question papers and explains each topic in simple language.

Author: kumar

Contents

MyCollegeVerse.in Branding

MyCollegeVerse.in

Building Academic Identity – The Student OS

MyCollegeVerse.in aims to give students clean, exam-focused resources that feel like a small book and not just loose notes. This document on "OOP with Java" is prepared in a simple layout so that it can be printed or read as a black and white PDF. You can use it along with your classroom notes, previous year question papers and online practice.

1 Introduction to Java and Object Oriented Programming

1.1 Overview and Exam Pattern Focus

In AKTU papers of Object Oriented Programming with Java, the first unit usually covers Java basics and core Object Oriented Programming ideas. Questions often ask definitions, features, and short programs. Many long answer questions repeat around topics like features of Java, the Java platform, and OOP principles. If you understand this unit clearly, it becomes easier to answer later units also.

1.2 Java Platform: JVM, JRE and JDK

1.2.1 Basic Definitions

- **JVM (Java Virtual Machine)** is the part that actually runs Java bytecode. It acts as a virtual computer that understands `.class` files and executes them.
- **JRE (Java Runtime Environment)** includes the JVM and libraries that are needed to run Java programs. If you only need to run existing Java applications, JRE is enough.
- **JDK (Java Development Kit)** includes the JRE plus development tools like `javac` compiler and debugging tools. If you want to write and compile Java programs, you use the JDK.

1.2.2 Diagram: Relationship Between JDK, JRE and JVM

This simple diagram is enough for most exam answers. In a long question, write the definitions and then draw this figure with labels.

1.3 Features of Java

Exams often ask: "Explain the main features of Java" or "Why is Java called platform independent and secure?" Some common features are:

- **Simple:** Java syntax is simpler than C++ because it removes pointers for direct memory access and multiple inheritance through classes.
- **Object Oriented:** Java treats almost everything as objects. Concepts like classes, objects, inheritance, and polymorphism are central to Java code.

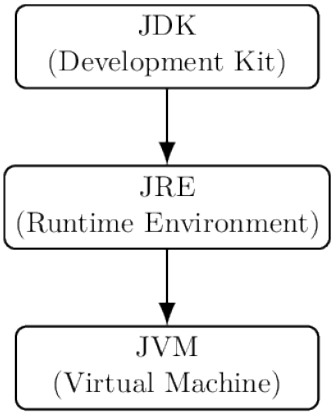

Figure 1.1: Simple relationship between JDK, JRE and JVM

- **Platform Independent:** Java programs are compiled into bytecode, which can run on any system that has a JVM. This is often called "write once, run anywhere".
- **Robust:** Java supports strong type checking, exception handling, and automatic garbage collection, which together reduce runtime errors.
- **Secure:** The absence of direct pointer arithmetic and the JVM sandbox model help in building secure applications.

1.4 Basic OOP Principles

1.4.1 Encapsulation

Encapsulation means combining data and methods that work on that data into a single unit called a class. The internal details are hidden from other classes using access modifiers like `private` and `public`. In exams, you can show a small class where some variables are private, and methods are used to access them.

1.4.2 Inheritance

Inheritance is a way to create a new class from an existing class. The new class (child) reuses the properties and methods of the old class (parent). This helps in code reusability and logical grouping of classes.

1.4.3 Polymorphism

Polymorphism means one name, many forms. In Java, methods can show different behaviour depending on the object that calls them or the parameters passed. Method overloading and method overriding are two common forms used in questions.

1.4.4 Abstraction

Abstraction hides complex details and only shows the necessary features. In Java, abstract classes and interfaces are used to define abstract behaviours that subclasses must implement.

2 Inheritance, Abstract Classes and Interfaces

2.1 Types of Inheritance in Java

From exam point of view, you should be able to describe different types of inheritance like single, multilevel and hierarchical. Java does not support multiple inheritance through classes to avoid ambiguity, but interfaces give a way to get similar behaviour.

- **Single inheritance:** One parent class and one child class.
- **Multilevel inheritance:** A class is derived from another derived class.
- **Hierarchical inheritance:** Multiple child classes are derived from one parent class.

2.2 Method Overloading and Method Overriding

2.2.1 Method Overloading

Method overloading happens when multiple methods in the same class have the same name but different parameter lists. It is decided at compile time.

2.2.2 Method Overriding

Method overriding occurs when a child class provides its own implementation of a method that is already defined in the parent class. It is resolved at run time and is directly linked to polymorphism.

2.3 Abstract Classes and Interfaces

2.3.1 Abstract Class

An abstract class is a class that cannot be instantiated. It can have abstract methods (methods without body) as well as normal methods. Exams often ask for definition, when to use abstract class, and a simple example.

2.3.2 Interface

An interface is a complete abstract type that defines a set of methods that a class must implement. It supports multiple inheritance behaviour because a class can implement multiple

interfaces.

2.3.3 Common Exam Question

A frequently asked question is: "Differentiate between abstract class and interface." In your answer, you can make a small table with 4–5 differences like object creation, method types, support for multiple inheritance, and default methods.

3 Exception Handling, Multithreading and Input/Output

3.1 Exception Handling

3.1.1 Basic Idea

An exception is an event that occurs during the execution of a program which disrupts the normal flow of instructions. Exception handling provides a safe way to handle such events and continue program execution in a controlled way.

3.1.2 Keywords Used

Important keywords that must be written in exam answers:

- **try:** Block of code where exception may occur.
- **catch:** Block that handles the exception.
- **finally:** Block that always executes, whether exception occurs or not.
- **throw:** Used to throw an exception explicitly.
- **throws:** Used in method signature to declare that a method may throw exceptions.

3.1.3 Checked and Unchecked Exceptions

Checked exceptions are checked at compile time (for example, `IOException`). Unchecked exceptions are not checked at compile time; they usually extend `RuntimeException` (for example, `NullPointerException`).

3.2 Multithreading

3.2.1 Thread and Process

A process is a running program. A thread is a smaller unit of a process that can run independently. In Java, we can create multiple threads within one program to perform tasks in parallel.

3.2.2 Thread Life Cycle

Exams often ask to "explain the life cycle of a thread with neat diagram". The main states are: New, Runnable, Running, Blocked/Waiting and Terminated.

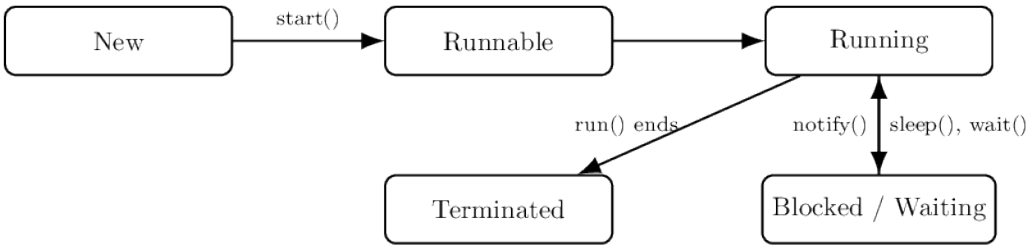

Figure 3.1: Simple life cycle of a thread in Java

3.2.3 Creating Threads

There are two standard ways to create a thread in Java:

1. Extending the `Thread` class.
2. Implementing the `Runnable` interface.

For exam programs, you should be able to write basic examples using both approaches.

3.3 Input/Output Streams

3.3.1 Byte and Character Streams

Java separates input and output into byte streams and character streams. Byte streams are used to handle raw binary data, while character streams are used for text.

3.3.2 Common Classes

- `FileInputStream` and `FileOutputStream` – for reading and writing bytes.
- `FileReader` and `FileWriter` – for reading and writing characters.
- `BufferedReader` – for efficient reading of text line by line.

A very common exam program is "Copy the contents of one file to another using streams". In your answer, show the main loop where bytes or characters are read from source and written to destination.

4 Collections Framework and Generics

4.1 Overview of Collections

The Collections Framework is a set of classes and interfaces that provide ready-made data structures like lists, sets and maps. Many recent AKTU papers include questions asking you to compare `ArrayList` and `LinkedList`, or `HashMap` and `TreeMap`.

4.2 Collections Hierarchy Diagram

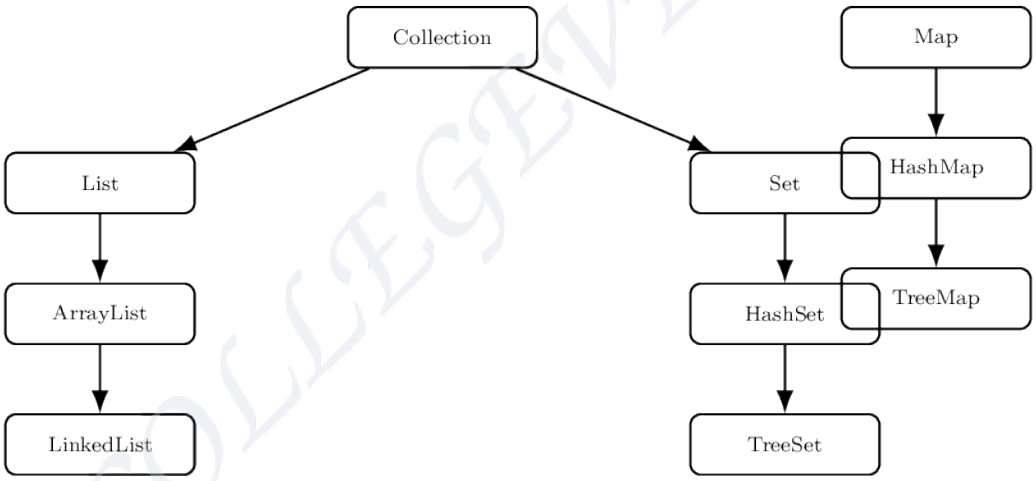

Figure 4.1: Simplified Collections Framework hierarchy

4.3 Important Comparisons

In exams, short answer questions often ask you to write differences between:

- `ArrayList` and `LinkedList`
- `HashSet` and `TreeSet`
- `HashMap` and `TreeMap` and `LinkedHashMap`

Use small comparison tables or 4–5 points each to answer these questions.

5 Modern Java Features and Revision Strategy

5.1 Functional Interfaces and Lambda Expressions

Recent question sets include basic questions on functional interfaces and lambda expressions.

- A **functional interface** is an interface with exactly one abstract method.
- A **lambda expression** is a short block of code that can be passed as data and matches the abstract method of a functional interface.

You should be able to define both terms and write one small example.

5.2 Simple Revision Plan

- First revise exception handling, multithreading and I/O streams, because these topics repeat most.
- Then revise core OOP principles and Java basics.
- After that, focus on inheritance, abstract classes and interfaces.
- Finally, revise the Collections Framework and modern Java features.

This completes the basic book-style layout for AKTU OOP with Java. You can add your own solved programs, previous year questions and extra diagrams on blank pages after each chapter if needed.