***Java –Assignment 2025***

***Module 1 – Core Java***

***1. Introduction to Java***

***Theory:***

***o History of Java :***

# ***Java's history began in 1991 at Sun Microsystems, with a team led by James Gosling initially developing a language called "Oak" for interactive television, later renamed to Java and released as Java 1.0 in 1996.***

***o Features of Java (Platform Independent, Object-Oriented, etc.)***

***The primary objective of*** [***Java programming***](https://www.tpointtech.com/java-tutorial) ***language creation was to make it portable, simple and secure programming language. Apart from this, there are also some excellent features which play an important role in the popularity of this language. The features of Java are also known as Java buzzwords.***

***A list of the most important features of the Java language is given below.***

[***Simple,***](https://www.tpointtech.com/features-of-java#Simple) [***Object-Oriented***](https://www.tpointtech.com/features-of-java#Object-Oriented) ***,*** [***Portable,***](https://www.tpointtech.com/features-of-java#Portable) [***Platform independent***](https://www.tpointtech.com/features-of-java#Platform-independent)

[***Secured,***](https://www.tpointtech.com/features-of-java#Secured) [***Robust,***](https://www.tpointtech.com/features-of-java#Robust) [***Architecture neutral,***](https://www.tpointtech.com/features-of-java#Architecture-neutral) [***Interpreted***](https://www.tpointtech.com/features-of-java#Interpreted)[***High Performance,***](https://www.tpointtech.com/features-of-java#High-Performance) ***Multithreaded, Distribute,*** [***Dynamic***](https://www.tpointtech.com/features-of-java#Dynamic)

***o Understanding JVM, JRE, and JDK:***

***JVM is stand for java virtual machine which is used for the compilation of the program. And JRE stand for java runtime environment which is used for run or execution of the programs. And JDK is a java development kit which provide users a various kind of inbuilt libraries or kit which help programming easy.***

***o Setting up the Java environment and IDE (e.g., Eclipse, IntelliJ):***

***We have various kind of platform where we use to work on java code they are Eclipse, Intellije IDE, Notepad, Notepad ++, Vs code etc .***

***o Java Program Structure (Packages, Classes, Methods) :***

***The syntax or structure for java program is:***

***package core java;***

***class Shiv*** ***{***

***public static void main (String [] args){***

***}***

***}***

***2. Data Types, Variables, and Operators***

***Theory:***

***o Primitive Data Types in Java (int, float, char, etc.)***

***In Java, primitive data types are the most basic types of data that are predefined by the language. They represent simple values and are not objects. Java has 8 primitive data types, and each one is used to store specific kinds of data.***

***Here are the 8 primitive data types in Java:***

1. ***byte:***
   1. ***Size: 1 byte (8 bits)***
   2. ***Range: -128 to 127***
   3. ***Used for saving memory in large arrays of integers.***
2. ***short:***
   1. ***Size: 2 bytes (16 bits)***
   2. ***Range: -32,768 to 32,767***
   3. ***Used for saving memory when you know the range of values will not exceed the short range.***
3. ***int:***
   1. ***Size: 4 bytes (32 bits)***
   2. ***Range: -2^31 to 2^31-1 (-2,147,483,648 to 2,147,483,647)***
   3. ***The most commonly used integer data type.***
4. ***long:***
   1. ***Size: 8 bytes (64 bits)***
   2. ***Range: -2^63 to 2^63-1***
   3. ***Used when an int is not large enough.***
5. ***float:***
   1. ***Size: 4 bytes (32 bits)***
   2. ***Range: 1.4E-45 to 3.4E+38 (approx.)***
   3. ***Used for decimal numbers with single precision.***
6. ***double:***
   1. ***Size: 8 bytes (64 bits)***
   2. ***Range: 4.9E-324 to 1.8E+308 (approx.)***
   3. ***Used for decimal numbers with double precision (more accurate than float).***
7. ***char:***
   1. ***Size: 2 bytes (16 bits)***
   2. ***Range: 0 to 65,535***
   3. ***Represents a single Unicode character.***
8. ***Boolean:***
   1. ***Size: 1 bit (but may vary depending on JVM implementation)***
   2. ***Values: true or false***
   3. ***Used for logical values.***

***o Variable Declaration and Initialization:***

***Variable Declaration Variable declaration is the process of specifying the name and data type of a variable.***

***Syntax:***

***data\_type variable\_name;***

***Example:***

***int myInteger; double myDouble;***

***Variable Initialization Variable initialization is the process of assigning a value to a variable.***

***Syntax:***

***data\_type variable\_name = value;***

***Example:***

***int myInteger = 10; double myDouble = 3.14;***

***o Operators: Arithmetic, Relational, Logical, Assignment, Unary, and Bitwise:***

***Operators***

***1. Arithmetic Operators:***

***- Add: +***

***- Subtract: -***

***- Multiply: \****

***- Divide: /***

***2. Relational Operators:***

***- Equal: ==***

***- Not Equal: !=***

***- Greater: >***

***- Less: <***

***3. Logical Operators:***

***- And: &&***

***- Or: ||***

***- Not: !***

***4. Assignment Operators:***

***- Assign: =***

***- Add and Assign: +=***

***- Subtract and Assign: -=***

***5. Unary Operators:***

***- Increment: ++***

***- Decrement: --***

***6. Bitwise Operators:***

***- AND: &***

***- OR: |***

***- XOR: ^***

***o Type Conversion and Type Casting:***

***Type Conversion***

***- Changing a value from one data type to another.***

***- Automatic conversion by Java (e.g., int to double).***

***Type Casting***

***- Manually changing a value from one data type to another.***

***- Using parentheses to specify the target data type (e.g., (int) double).***

***Example:***

***- Type Conversion: double x = 10; (int to double)***

***- Type Casting: int y = (int) 10.5; (double to int)***

***3. Control Flow Statements***

***Theory:***

***o If-Else Statements:***

***f-Else Statement***

***- Used for decision-making in code.***

***- Checks a condition and executes different blocks of code.***

***If:***

***- Check a condition.***

***- If true, execute the code inside the "if" block.***

***Else:***

***- If the condition is false, execute the code inside the "else" block.***

***o Switch Case Statements:***

***Switch Case***

***- Used for decision-making in code.***

***- Checks a value and executes different blocks of code.***

***How it works:***

***1. The "switch" statement checks the value of a variable.***

***2. The value is compared to each "case" value.***

***3. If a match is found, the code inside that "case" block is executed.***

***o Loops (For, While, Do-While):***

***oops***

***- Used to repeat a block of code.***

***- Continues to execute until a condition is met.***

***Types of Loops:***

***1. For Loop: Used to iterate a fixed number of times.***

***Example: for (int i = 0; i < 5; i++) { ... }***

***2. While Loop: Used to iterate while a condition is true.***

***Example: int i = 0; while (i < 5) { ...; i++; }***

***3. Do-While Loop: Used to iterate at least once, then while a condition is true.***

***Example: int i = 0; do { ...; i++; } while (i < 5);***

***o Break and Continue Keywords:***

***Break***

***- Used to exit a loop or switch statement immediately.***

***- Transfers control to the next statement after the loop or switch.***

***Continue***

***- Used to skip the current iteration of a loop.***

***- Moves control to the next iteration of the loop.***

***4. Classes and Objects:***

***Classes***

***- A blueprint or template for creating objects.***

***- Defines properties (data) and methods (functions).***

***- A class is essentially a design pattern or a template.***

***Objects***

***- An instance of a class.***

***- Has its own set of attributes (data) and methods (functions).***

***- Represents a real-world entity or concept.***

***Theory:***

***o Defining a Class and Object in Java***

***Class (Java)***

***A class in Java is a blueprint or template that defines the properties and behavior of an object. It is defined using the class keyword and typically includes:***

***- Variables (data members)***

***- Methods (functions)***

***- Constructors***

***Object (Java)***

***An object in Java is an instance of a class, and has its own set of attributes (data) and methods (functions). It is created using the new keyword.***

***Example:***

***Car myCar = new Car("Red", 60);***

***o Constructors and Overloading:***

***Constructor***

***- A special method in a class that is called when an object is created.***

***- Used to initialize the object's properties.***

***- Has the same name as the class.***

***- No return type, not even void.***

***Constructor Overloading***

***- Having multiple constructors in a class with different parameter lists.***

***- Allows objects to be created with different sets of properties.***

***o Object Creation, Accessing Members of the Class:***

***Object Creation***

***- Creating an instance of a class using the new keyword.***

***- Allocates memory for the object.***

***Example:***

***Car myCar = new Car("Red", 60);***

***Accessing Members of the Class***

***- Accessing variables (data members) and methods of a class using the dot (.) operator.***

***- Can be used to read or modify variable values or call methods.***

***Note:***

***- myCar is the object reference.***

***- color and speed are variables (data members).***

***- accelerate() is a method.***

***o this Keyword:***

***This Keyword***

***- A reference to the current object of the class.***

***- Used to access class members (variables and methods) from within the class.***

***- Can be used to:***

***5. Methods in Java***

***Theory:***

***o Defining Methods:***

***Method***

***A method is a block of code that:***

***1. Performs a specific task***

***2. Can be called multiple times from different parts of a program***

***3. Can accept parameters (inputs) and return values (outputs)***

***Characteristics of a Method:***

***1. Name: A unique identifier for the method***

***2. Parameters: Inputs passed to the method***

***3. Return Type: The data type of the value returned by the method***

***4. Body: The code that executes when the method is called***

***o Method Parameters and Return Types:***

***Method Parameters***

***- Values passed to a method when it's called.***

***- Listed inside the method's parentheses.***

***- Can be primitive types (int, double, etc.) or reference types (objects, arrays, etc.).***

***In this example:***

***- name is a parameter of type String.***

***Return Type***

***- The data type of the value returned by a method.***

***- Specified before the method name.***

***- Can be primitive types (int, double, etc.) or reference types (objects, arrays, etc.).***

***o Method Overloading:***

***Method Overloading***

***Method overloading is a feature in Java that allows:***

***1. Multiple methods with the same name***

***2. Different parameter lists (number, type, or order)***

***Rules for Method Overloading:***

***1. Method name must be same***

***2. Parameter list must be different***

***3. Return type can be different (but not required)***

***o Static Methods and Variables :***

***Static Method***

***A static method is a method that:***

***1. Belongs to a class, not an instance***

***2. Can be called without creating an object***

***3. Has no access to instance variables (non-static variables)***

***Static Variable***

***A static variable is a variable that:***

***1. Belongs to a class, not an instance***

***2. Is shared by all instances of the class***

***3. Is initialized only once, when the class is loaded***

***Key Points:***

***1. Static methods can only access static variables***

***2. Static variables are initialized only once***

***3. Static methods and variables are essentially global***

***6. Object-Oriented Programming (OOPs) Concepts***

***Theory:***

***o Basics of OOP: Encapsulation, Inheritance, Polymorphism, Abstraction:***

***Encapsulation***

***- Hiding internal details: Concealing the implementation details of an object from the outside world.***

***- Showing only necessary information: Exposing only the necessary information to the outside world through public methods.***

***- Protecting data: Protecting the data from external interference and misuse.***

***Inheritance***

***- Creating a new class from an existing class: Creating a new class that inherits the properties and behavior of an existing class.***

***- Code reusability: Reusing the code of the existing class.***

***- Hierarchical relationship: Creating a hierarchical relationship between classes.***

***Polymorphism***

***- Many forms: The ability of an object to take on multiple forms.***

***- Method overriding: When a subclass provides a different implementation of a method that is already defined in its superclass.***

***- Method overloading: When multiple methods with the same name can be defined with different parameter lists.***

***Abstraction***

***- Showing only essential features: Showing only the essential features of an object or system while hiding the internal details.***

***- Focusing on interfaces: Focusing on the interfaces and interactions between objects rather than their internal implementation.***

***- Simplifying complex systems: Simplifying complex systems by exposing only the necessary information.***

***o Inheritance: Single, Multilevel, Hierarchical:***

***Single Inheritance***

***- A child class inherits from a single parent class.***

***- The child class has access to all the members of the parent class.***

***Multilevel Inheritance***

***- A child class inherits from a parent class, which in turn inherits from another parent class.***

***- The child class has access to all the members of all the parent classes.***

***Hierarchical Inheritance***

***- A parent class is inherited by multiple child classes.***

***- Each child class has access to all the members of the parent class.***

***o Method Overriding and Dynamic Method Dispatch:***

***Method Overriding***

***- A subclass provides a different implementation of a method that is already defined in its superclass.***

***- The method name, return type, and parameter list must be exactly the same.***

***- Allows for more specific behavior in the subclass.***

***Dynamic Method Dispatch***

***- The decision of which method to call is made at runtime, not compile-time.***

***- The actual method called depends on the type of object being referred to, not the type of reference variable.***

***- Allows for more flexibility and polymorphism in programming.***

***7. Constructors and Destructors***

***Theory:***

***o Constructor Types (Default, Parameterized):***

1. ***Default Constructor***

***-No parameters***

***-Used to initialize objects with default values***

***-If no constructor is defined, the compiler provides a default constructor***

1. ***Parameterized Constructor***

***-One or more parameters***

***-Used to initialize objects with specific values***

1. ***Copy Constructor***

***-A constructor that creates a copy of an existing object***

***-Typically used to create a duplicate of an object***

1. ***Private Constructor***

***-A constructor that is declared private***

***-Used to prevent object creation from outside the class***

***o Copy Constructor (Emulated in Java):***

***Copy Constructor A copy constructor is a constructor that creates a copy of an existing object.***

***Emulation in Java Java does not support copy constructors in the classical sense. However, you can emulate a copy constructor using various techniques:***

1. ***Cloneable interface: Implement the Cloneable interface and override the clone() method to create a copy of the object.***
2. ***Copy constructor-like method: Create a static method that takes an object as a parameter and returns a new object with the same state.***
3. ***Constructor with object parameter: Create a constructor that takes an object of the same class as a parameter and initializes the new object with the same state.***

***o Constructor Overloading:***

***Constructor Overloading Constructor overloading is a technique in which multiple constructors with different parameter lists are defined in a class.***

***Rules for Constructor Overloading:***

1. ***Multiple constructors: A class can have multiple constructors.***
2. ***Different parameter lists: Each constructor must have a unique parameter list.***
3. ***Same constructor name: All constructors must have the same name as the class.***
4. ***No return type: Constructors do not have a return type.***

***Benefits of Constructor Overloading:***

1. ***Increased flexibility: Multiple constructors provide more flexibility in object creation.***
2. ***Improved code readability: Constructor overloading makes the code more readable by providing multiple ways to create an object.***

***o Object Life Cycle and Garbage Collection:***

***Object Life Cycle***

***1. Creation: An object is created using the new keyword.***

***2. Initialization: The object's constructor is called to initialize its state.***

***3. Usage: The object is used by the program.***

***4. Dormancy: The object is no longer referenced or used.***

***5. Garbage Collection: The object is reclaimed by the garbage collector.***

***Garbage Collection***

***1. Mark: The garbage collector identifies objects that are no longer referenced.***

***2. Sweep: The garbage collector reclaims the memory occupied by the marked objects.***

***3. Compact: The garbage collector rearranges the remaining objects to remove any gaps.***

***Types of Garbage Collection***

***1. Minor GC: Collects young objects (short-lived objects).***

***2. Major GC: Collects old objects (long-lived objects).***

***3. Full GC: Collects all objects (young and old).***

***Benefits of Garbage Collection***

***1. Memory Management: Automatic memory management eliminates memory leaks.***

***2. Reduced Errors: No need to worry about freeing memory.***

***3. Improved Performance: Garbage collection optimizes memory usage.***

***8. Arrays and Strings***

***Theory:***

***o One-Dimensional and Multidimensional Arrays:***

***1D Array (Single-Dimensional Array)***

***- A collection of elements of the same data type stored in contiguous memory locations.***

***- Each element is identified by a single index or subscript.***

***Multi-Dimensional Array***

***- An array of arrays, where each element is itself an array.***

***- Each element is identified by multiple indices or subscripts.***

***Types of Multi-Dimensional Arrays:***

***1. 2D Array (Two-Dimensional Array)***

***- An array of arrays, where each element is an array of a fixed size.***

***2. 3D Array (Three-Dimensional Array)***

***- An array of 2D arrays, where each element is a 2D array.***

***3. Jagged Array (Irregular Array)***

***- An array of arrays, where each element is an array of varying size.***

***o String Handling in Java: String Class, StringBuffer, StringBuilder:***

***tring Class***

***- Immutable (cannot be changed once created)***

***- Thread-safe***

***- Methods:***

***- concat()***

***- equals()***

***- length()***

***- substring()***

***- toUpperCase()***

***- toLowerCase()***

***StringBuffer***

***- Mutable (can be changed after creation)***

***- Thread-safe***

***- Methods:***

***- append()***

***- insert()***

***- delete()***

***- reverse()***

***StringBuilder***

***- Mutable (can be changed after creation)***

***- Not thread-safe***

***- Methods:***

***- append()***

***- insert()***

***- delete()***

***- reverse()***

***o Array of Objects:***

***Array of Objects***

***- An array that stores objects as its elements.***

***- Each element in the array is an object, which can be of the same class or different classes.***

***Declaring an Array of Objects***

***- ClassName[] arrayName = new ClassName[size];***

***Initializing an Array of Objects***

***- arrayName[0] = new ClassName();***

***- arrayName[1] = new ClassName();***

***Accessing Elements of an Array of Objects***

***- arrayName[0].methodName();***

***Benefits of Arrays of Objects***

***- Grouping related objects: Store multiple objects of the same class or different classes in a single array.***

***- Easier manipulation: Perform operations on multiple objects using loops.***

***Example Use Cases***

***- Student database: Store student objects in an array to manage student data.***

***- Product inventory: Store product objects in an array to manage product inventory.***

***o String Methods (length, charAt, substring, etc.) :***

1. ***length()***

***-Returns the length of the string.***

1. ***charAt()***

***-Returns the character at the specified index.***

1. ***substring()***

***-Returns a substring of the original string.***

***-Can take one or two parameters: start index, and optional end index.***

1. ***concat()***

***-Concatenates two strings together.***

1. ***indexOf()***

***-Returns the index of the first occurrence of the specified character or substring.***

1. ***lastIndexOf()***

***-Returns the index of the last occurrence of the specified character or substring.***

1. ***toUpperCase()***

***-Returns a new string with all characters converted to uppercase.***

1. ***toLowerCase()***

***-Returns a new string with all characters converted to lowercase.***

***9. Inheritance and Polymorphism:***

***Theory:***

***o Inheritance Types and Benefits:***

1. ***Single Inheritance***

***-A child class extends a single parent class.***

1. ***Multilevel Inheritance***

***- child class extends a parent class, which in turn extends another parent class.***

1. ***Hierarchical Inheritance***

***-Multiple child classes extend a single parent class.***

1. ***Note:***

***-Java does not support multiple inheritance of state.***

***-Java does support multiple inheritance of behavior through interfaces.***

***o Method Overriding:***

***- Method overriding is a feature of object-oriented programming (OOP) that allows a subclass to provide a different implementation of a method that is already defined in its superclass.***

***o Dynamic Binding (Run-Time Polymorphism):***

* ***Dynamic binding, also known as run-time polymorphism, is a feature of object-oriented programming (OOP) that allows the correct method to be called at run-time, based on the type of object being referred to.***

***o Super Keyword and Method Hiding:***

***super Keyword***

***- The super keyword is used to access the members (methods and fields) of a superclass.***

***- It is used to call the constructor of a superclass.***

***- It is used to access the methods and fields of a superclass that are hidden by a subclass.***

***Method Hiding***

***- Method hiding occurs when a subclass provides a method with the same name and signature as a method in its superclass.***

***- The method in the subclass hides the method in the superclass.***

***- To access the hidden method, the super keyword can be used.***

***10. Interfaces and Abstract Classes***

***Theory:***

***o Abstract Classes and Methods:***

***Abstract Class***

***- An abstract class is a class that cannot be instantiated on its own.***

***- It is designed to be inherited by other classes.***

***- An abstract class can have both abstract and non-abstract methods.***

***- A class that extends an abstract class must provide an implementation for all abstract methods.***

***Abstract Method***

***- An abstract method is a method that is declared without an implementation.***

***- It is declared with the abstract keyword.***

***- A subclass must provide an implementation for an abstract method.***

***- An abstract method can only be declared in an abstract class.***

***o Interfaces: Multiple Inheritance in Java:***

***Interfaces***

***- An interface is a abstract class that contains only abstract methods and constants.***

***- Interfaces are used to define a contract or a set of methods that must be implemented by any class that implements it.***

***- Interfaces are declared using the interface keyword.***

***Multiple Inheritance***

***- Java does not support multiple inheritance of state (i.e., a class cannot extend multiple classes).***

***- However, Java does support multiple inheritance of behavior through interfaces (i.e., a class can implement multiple interfaces).***

***o Implementing Multiple Interfaces :***

***Multiple interface implementation: A class can implement multiple interfaces by listing them after the implements keyword.***

***- Interface methods must be implemented: A class implementing multiple interfaces must provide an implementation for all methods declared in the interfaces.***

***- Interface methods can be overridden: A class implementing multiple interfaces can override methods declared in the interfaces.***

***11. Packages and Access Modifiers***

***Theory:***

***o Java Packages: Built-in and User-Defined Packages:***

***A package is a collection of related classes, interfaces, and other types.***

***- Packages are used to organize and structure Java code.***

***Built-in Packages***

***- Java has several built-in packages that provide a wide range of classes and interfaces for tasks such as:***

***- Input/Output (java.io)***

***- Networking (java.net)***

***- Collections (java.util)***

***- Graphics (java.awt)***

***o Access Modifiers: Private, Default, Protected, Public:***

***Access modifiers are keywords that determine the accessibility of a class, method, or variable.***

***They control who can access and modify the data and behavior of a class.***

***Types of Access Modifiers***

* ***Public: Accessible from anywhere, both within and outside the class.***
* ***Private: Accessible only within the same class.***
* ***Protected: Accessible within the same class and subclasses.***
* ***Default (no modifier): Accessible within the same package.***

***o Importing Packages and Classpath:***

***Importing Packages***

***- In Java, you can import packages using the import statement.***

***- The import statement allows you to use classes and interfaces from other packages without having to qualify them with the full package name.***

***Types of Import Statements***

***1. Specific Import: Import a specific class or interface from a package.***

***Example: import java.util.ArrayList;***

***2. Wildcard Import: Import all classes and interfaces from a package.***

***Example: import java.util.\*;***

***Classpath***

***- The classpath is the path where the Java Virtual Machine (JVM) looks for classes and packages.***

***- The classpath can be set using the -cp or -classpath option when running the Java program.***

***12. Exception Handling***

***Theory:***

***o Types of Exceptions: Checked and Unchecked***

***1. Checked Exceptions***

***These are exceptions that are checked at compile-time. The compiler ensures that these exceptions are handled using try-catch blocks or declared using the throws keyword in the method signature.***

#### ***Examples of Checked Exceptions***

* ***IOException – When dealing with file handling or network operations.***
* ***SQLException – When dealing with databases.***
* ***FileNotFoundException – When trying to access a file that does not exist.***

***2. Unchecked Exceptions***

***These are exceptions that occur during runtime and are not checked at compile-time. They usually indicate programming logic errors, such as dividing by zero or accessing an invalid array index.***

***Examples of Unchecked Exceptions***

***NullPointerException – When trying to access an object that is null.***

***ArithmeticException – When dividing by zero.***

***ArrayIndexOutOfBoundsException – When accessing an invalid index of an array.***

***o try, catch, finally, throw, throws***

***1. try***

* ***Defines a block of code where exceptions might occur.***
* ***Must be followed by either catch or finally.***

#### ***2. catch***

* ***Handles exceptions thrown in the try block.***
* ***Can have multiple catch blocks for different exception types.***

#### ***3. finally***

* ***Always executes after try and catch, regardless of whether an exception occurred.***
* ***Used for resource cleanup (e.g., closing files or database connections).***

#### ***4. throw***

* ***Used to explicitly throw an exception.***
* ***Can be used inside methods or conditionally based on logic.***

#### ***5. throws***

* ***Declares exceptions that a method might throw.***
* ***Used in method signatures to indicate that exceptions need to be handled where the method is called.***

***o Custom Exception Classes***

***Custom Exception Handling in Java***

***Custom exceptions allow developers to define their own exception classes, extending Java's built-in Exception or RuntimeException classes. This helps in creating meaningful and application-specific error messages.***

## ***1. Creating a Custom Checked Exception***

* ***Extend the Exception class.***
* ***Requires explicit handling using try-catch or throws.***

### ***Steps:***

1. ***Create a class extending Exception.***
2. ***Define constructors for custom messages.***
3. ***Use throw to trigger the exception.***
4. ***Handle it using try-catch or throws.***

## ***2. Creating a Custom Unchecked Exception***

* ***Extend the RuntimeException class.***
* ***Does not require explicit handling.***

### ***Steps:***

1. ***Create a class extending RuntimeException.***
2. ***Define constructors for custom messages.***
3. ***Use throw to trigger the exception.***

***13. Multithreading***

***Theory:***

***o Introduction to Threads***

***A thread is the smallest unit of execution in a program. Java provides built-in support for multithreading, which allows multiple tasks to run concurrently, improving performance and responsiveness.***

***o Creating Threads by Extending Thread Class or Implementing Runnable Interface***

***Java provides two ways to create threads:***

#### ***1. Extending the Thread Class***

* ***Create a class that extends Thread.***
* ***Override the run() method to define the task.***
* ***Create an object of the class and call start() to begin execution.***
* ***Not recommended if the class needs to extend another class (since Java doesn’t support multiple inheritance).***

#### ***2. Implementing the Runnable Interface***

* ***Create a class that implements Runnable.***
* ***Override the run() method to define the task.***
* ***Pass an instance of this class to a Thread object.***
* ***More flexible as it allows the class to extend other classes.***

***o Thread Life Cycle***

***A thread in Java goes through five stages in its life cycle:***

***1️⃣ New (Created)***

***The thread is created but not yet started.***

***It is in this state after calling the Thread constructor.***

***2️⃣ Runnable (Ready to run)***

***The thread is ready to run but waiting for CPU time.***

***It enters this state after calling start().***

***3️⃣ Running (Executing)***

***The thread is actively executing its task.***

***The JVM scheduler decides when the thread runs.***

***4️⃣ Blocked / Waiting / Timed Waiting***

***The thread is paused, waiting for resources or time.***

***Happens when a thread waits for another thread to complete or for a lock.***

***5️⃣ Terminated (Dead)***

***The thread has completed execution or was stopped.***

***It cannot be restarted once terminated.***

***o Synchronization and Inter-thread Communication***

***1. Synchronization***

***Synchronization ensures that multiple threads do not access shared resources simultaneously, preventing data inconsistency.***

#### ***Types of Synchronization:***

***✔ Synchronized Method – Locks an entire method.***  
 ***✔ Synchronized Block – Locks only a specific block of code.***  
 ***✔ Static Synchronization – Locks a class-level resource.***

***When to use?***

* ***When multiple threads modify shared data.***
* ***To avoid race conditions and data inconsistency.***

### ***2. Inter-thread Communication***

***Inter-thread communication allows threads to cooperate by waiting and notifying each other.***

#### ***Key Methods for Inter-thread Communication:***

***✔ wait() – Makes a thread wait until notified.***  
 ***✔ notify() – Wakes up a single waiting thread.***  
 ***✔ notifyAll() – Wakes up all waiting threads.***

***When to use?***

* ***In producer-consumer problems where one thread produces data, and another consumes it.***
* ***When threads need to wait for specific conditions before proceeding.***

***14. File Handling***

***Theory:***

***o Introduction to File I/O in Java (java.io package)***

***Introduction to File I/O in Java (java.io Package)***

***File I/O (Input/Output) in Java allows reading from and writing to files using the java.io package. It is used for handling files, streams, and other input/output operations.***

### ***1. Key Classes in java.io Package***

#### ***For Reading Files (Input Streams)***

***✔ FileReader – Reads character files.***  
 ***✔ BufferedReader – Efficiently reads text using a buffer.***  
 ***✔ FileInputStream – Reads binary files.***

#### ***For Writing Files (Output Streams)***

***✔ FileWriter – Writes character files.***  
 ***✔ BufferedWriter – Writes text efficiently using a buffer.***  
 ***✔ FileOutputStream – Writes binary files.***

#### ***For File Management***

***✔ File – Represents a file or directory, allowing operations like create, delete, and check properties.***

### ***2. File Handling Operations***

***✔ Reading a file – Retrieving data from a file.***  
 ***✔ Writing to a file – Storing data in a file.***  
 ***✔ Appending to a file – Adding new content without overwriting existing data.***  
 ***✔ Checking file existence – Verifying if a file exists before performing operations.***

***o FileReader and FileWriter Classes***

***Introduction to File I/O in Java (java.io Package)***

***File I/O (Input/Output) in Java allows reading from and writing to files using the java.io package. It is used for handling files, streams, and other input/output operations.***

### ***1. Key Classes in java.io Package***

#### ***For Reading Files (Input Streams)***

***✔ FileReader – Reads character files.***  
 ***✔ BufferedReader – Efficiently reads text using a buffer.***  
 ***✔ FileInputStream – Reads binary files.***

#### ***For Writing Files (Output Streams)***

***✔ FileWriter – Writes character files.***  
 ***✔ BufferedWriter – Writes text efficiently using a buffer.***  
 ***✔ FileOutputStream – Writes binary files.***

#### ***For File Management***

***✔ File – Represents a file or directory, allowing operations like create, delete, and check properties.***

### ***2. File Handling Operations***

***✔ Reading a file – Retrieving data from a file.***  
 ***✔ Writing to a file – Storing data in a file.***  
 ***✔ Appending to a file – Adding new content without overwriting existing data.***  
 ***✔ Checking file existence – Verifying if a file exists before performing operations.***

***o BufferedReader and BufferedWriter***

***The BufferedReader and BufferedWriter classes are used for efficient reading and writing of character-based files by using an internal buffer.***

### ***1. BufferedReader (Reading a File Efficiently)***

* ***Wraps FileReader for efficient reading.***
* ***Reads text line by line instead of character by character.***
* ***Uses an internal buffer to improve performance.***

### ***2. BufferedWriter (Writing to a File Efficiently)***

* ***Wraps FileWriter for efficient writing.***
* ***Writes text line by line instead of character by character.***
* ***Uses an internal buffer to improve performance.***
* ***Supports flush() to ensure data is written immediately.***

***o Serialization and Deserialization***

***Serialization and deserialization are used to convert objects into a stream of bytes and then reconstruct them later.***

### ***1. Serialization (Saving Objects to a File/Stream)***

* ***Converts an object into a byte stream.***
* ***Used to save object states or transmit data over a network.***
* ***Implemented using ObjectOutputStream and Serializable interface.***

### ***2. Deserialization (Restoring Objects from a File/Stream)***

* ***Converts a byte stream back into an object.***
* ***Restores saved object states.***
* ***Implemented using ObjectInputStream.***

***15. Collections Framework***

***Theory:***

***o Introduction to Collections Framework***

***The Java Collection Framework provides a set of classes and interfaces for storing and manipulating groups of objects efficiently. It includes lists, sets, queues, and maps to handle different data structures.***

***o List, Set, Map, and Queue Interfaces***

***1. List Interface (java.util.List)***

* ***Ordered collection that allows duplicates.***
* ***Elements are accessed using index positions.***
* ***Supports random access (fast retrieval).***

## ***2. Set Interface (java.util.Set)***

* ***Unordered collection that does not allow duplicates.***
* ***Does not allow indexed access.***

## ***3. Map Interface (java.util.Map)***

* ***Stores key-value pairs (keys must be unique).***
* ***Does not extend Collection (separate interface).***

## ***4. Queue Interface (java.util.Queue)***

* ***FIFO (First In, First Out) structure.***
* ***Used for task scheduling and buffering.***

***o ArrayList, LinkedList, HashSet, TreeSet, HashMap, TreeMap***

## ***1. ArrayList (Dynamic Array - List Interface)***

***✔ Ordered, allows duplicates.***  
 ***✔ Fast random access (index-based).***  
 ***✔ Slow for insertions/deletions in the middle.***

## ***2. LinkedList (Doubly Linked List - List Interface)***

***✔ Ordered, allows duplicates.***  
 ***✔ Fast insertions/deletions (no shifting needed).***  
 ***✔ Slow random access (traverses from start).***

## ***3. HashSet (Unordered Set - Set Interface)***

***✔ Unordered, does not allow duplicates.***  
 ***✔ Uses hashing, making it fast for lookups (O(1)).***  
 ***✔ No guarantees on element order.***

## ***4. TreeSet (Sorted Set - Set Interface)***

***✔ Sorted, does not allow duplicates.***  
 ***✔ Uses a Red-Black Tree for ordering.***  
 ***✔ Slower than HashSet (O(log n) operations).***

## ***5. HashMap (Unordered Key-Value Store - Map Interface)***

***✔ Key-value pairs, keys must be unique.***  
 ***✔ Fast lookups (O(1)) using hashing.***  
 ***✔ No guarantees on order.***

## ***6. TreeMap (Sorted Key-Value Store - Map Interface)***

***✔ Sorted key-value pairs (keys must be unique).***  
 ***✔ Uses Red-Black Tree for ordering (O(log n)).***  
 ***✔ Slower than HashMap but maintains key order.***

***o Iterators and ListIterators***

***Iterators are used to traverse elements in collections like List, Set, and Map. Java provides two main iterator types:***

***1. Iterator (Universal Iterator)***

***✔ Works with all Collection types (List, Set, Queue).***  
 ***✔ Provides forward-only traversal.***  
 ***✔ Methods:***

***hasNext() → Checks if more elements exist.***

***next() → Returns the next element.***

***remove() → Removes the current element.***

***2. ListIterator (For Lists Only)***

***✔ Works only with Lists (ArrayList, LinkedList).***  
 ***✔ Supports both forward and backward traversal.***  
 ***✔ Can modify elements during iteration.***  
 ***✔ Methods (in addition to Iterator methods):***

***hasPrevious() → Checks if there’s a previous element.***

***previous() → Moves backward in the list.***

***add(E e) → Adds an element at the current position.***

***set(E e) → Updates the last returned element.***

***16. Java Input/Output (I/O)***

***Theory:***

***o Streams in Java (InputStream, OutputStream)***

***Java streams are used to perform input and output (I/O) operations in a byte-oriented manner. The java.io package provides two main types of streams:***

## ***1. InputStream (Reading Data)***

***✔ Abstract class for reading bytes from a source.***  
 ***✔ Used for reading files, keyboard input, and network data.***  
 ***✔ Common implementations:***

* ***FileInputStream → Reads bytes from a file.***
* ***BufferedInputStream → Improves performance with buffering.***
* ***ByteArrayInputStream → Reads bytes from an array.***

## ***2. OutputStream (Writing Data)***

***✔ Abstract class for writing bytes to a destination.***  
 ***✔ Used for writing to files, console, and network.***  
 ***✔ Common implementations:***

* ***FileOutputStream → Writes bytes to a file.***
* ***BufferedOutputStream → Improves performance with buffering.***
* ***ByteArrayOutputStream → Writes bytes to an array.***

***o Reading and Writing Data Using Streams***

***Streams in Java are used to read and write data in a byte-oriented manner.***

***1. Reading Data Using InputStream***

***Used to read bytes from a file or input source.***

***Common classes:***  
 ***✔ FileInputStream – Reads from a file.***  
 ***✔ BufferedInputStream – Improves efficiency by buffering data.***  
 ***✔ ByteArrayInputStream – Reads from a byte array.***

***2. Writing Data Using OutputStream***

***Used to write bytes to a file or output destination.***

***Common classes:***  
 ***✔ FileOutputStream – Writes to a file.***  
 ***✔ BufferedOutputStream – Buffers data before writing.***  
 ***✔ ByteArrayOutputStream – Writes to a byte array.***

***o Handling File I/O Operations***

***File I/O (Input/Output) operations in Java allow reading, writing, and managing files efficiently.***

***1. Common File I/O Classes***

***For Reading Files (Input)***

***✔ FileReader – Reads character files.***  
 ***✔ BufferedReader – Reads files efficiently line by line.***  
 ***✔ FileInputStream – Reads binary files (e.g., images, PDFs).***

***For Writing Files (Output)***

***✔ FileWriter – Writes character data to files.***  
 ***✔ BufferedWriter – Writes files efficiently with buffering.***  
 ***✔ FileOutputStream – Writes binary files.***

***For File Management***

***✔ File – Creates, deletes, checks properties of files and directories.***

***2. Key File I/O Operations***

***✔ Creating a File – Using File.createNewFile().***  
 ***✔ Checking File Existence – Using File.exists().***  
 ***✔ Reading a File – Using BufferedReader or FileInputStream.***  
 ***✔ Writing/Appending to a File – Using FileWriter or FileOutputStream.***  
 ***✔ Deleting a File – Using File.delete().***