Java Mastery Course

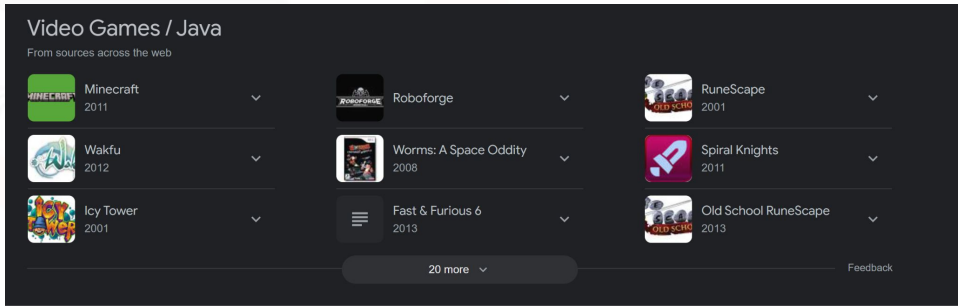
What is Java ?

- Java is a **programming language** and **platform**
- Java is a high level, robust, object-oriented and secure programming language.
- More than **3 billion** devices run Java.

ORACLE®

Where java is being used ?

1. Desktop applications
    a. Console based
    b. GUI based(AWT,Swing,JavaFX)
2. Mobile applications (specially **Android apps**)
3. Web applications (J2EE)
4. In Enterprise Huge Market (Spring Boot)
5. Games
6. And much, much more!



Why to learn Java ?

1. Java is one of the most popular programming language in market
2. Huge market demand
3. Work in different platform(windows, mac, linux, raspberry pi)
4. Open source and free
5. Secure , fast and powerful
6. Huge community (10 million developers)
7. Java is an object oriented language which gives a clear structure to programs.
8. If you learn java switch to other language is very easy

History of Java

1. Developed by **James Gosling** in the early 1990s
2. **James Gosling**, **Mike Sheridan**, and **Patrick Naughton** initiated the Java language project in June **1991**.
3. This small team of sun engineers called **Green Team**.
4. Initially it was designed for small, embedded systems in electronic appliances like set-top boxes
5. Firstly, it was called "**Greentalk**" by James Gosling, and the file extension was **.gt**
6. After that, it was called **Oak , Why Oak**
7. In 1995, Oak was renamed as "**Java**" because it was already a trademark by **Oak Technologies.**

Versions of Java

1. JDK Beta **1995**
2. JDK 1.0 January 23, **1996**
3. JDK 1.1 February 19**, 1997**
4. J2SE 1.2 December 8, **1998**
5. J2SE 1.3 May 8, **2000**
6. J2SE 1.4 February 6, **2002**
7. J2SE 5.0 September 30, **2004**
8. Java SE 6 December 11, **2006**
9. Java SE 7 July 28, **2011**
10. Java SE 8 (**LTS**) March 18, **2014**
11. Java SE 9 September 21, **2017**

Versions of Java

12. Java SE 10 March 20, **2018**
13. Java SE 11 (LTS) September 25, **2018**
14. Java SE 12 March 19, **2019**
15. Java SE 13 September 17, **2019**
16. Java SE 14 March 17, **2020**
17. Java SE 15 September 15, **2020**
18. Java SE 16 March 16, **2021**
19. Java SE 17 (LTS) September 14, **2021**
20. Java SE 18 March 22, **2022**
21. Java SE 19 September 20, **2022**
22. Java SE 20 March 21, 2023
23. Java SE 21 (LTS) September 19, **2023**

Features of Java

1. **Simple**
    a. Syntax is based on c,c++
    b. Removed complex concesp like **explicit pointers,operator overloading**
    c. Concepts like garbage collector. No need to free variables explicitly
2. **Platform Independent**: Write once run anywhere
3. **Secured**
    a. No explicit pointer
    b. Program run inside JVM sandbox
    c. Classloader
    d. Bytecode verifier
    e. Security Manager

Features of Java

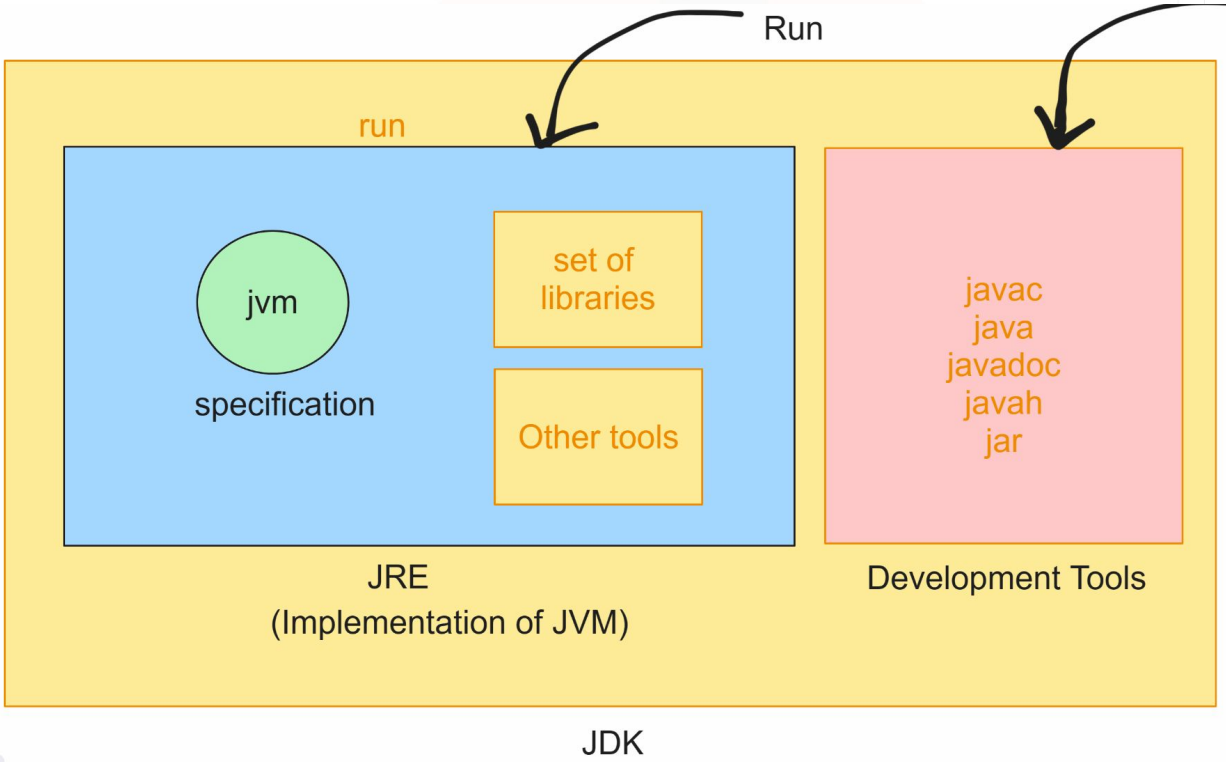
4. **Robust:** strong
    a. Exception Handling
    b. Strong memory management
    c. Garbage collection
5. **Object-oriented:** Everything in Java is an object. Program in java is composed of objects
    a. Inheritance
    b. Polymorphism
    c. Abstraction
    d. Encapsulation

Features of Java

6. Architecture-neutral
7. Portable
8. High-performance
9. Distributed
10. Multi-threaded
11. Dynamic

Lets setup java in local system

JVM, JRE , JDK



First Java Program

1. Editor/IDE
    a. Notepad
    b. Sublime
    c. VSCode
    d. Eclipse
    e. IntelliJ IDEA
    f. Netbeans
2. JDK (Develop and run programs)

Important Points

1. Can filename and class name be different
    a. Yes
    b. In case class is **public,** in that case class name and filename must be **same** otherwise compiler will complain.
2. Can we have multiple classes in single java file
    a. Yes
3. Can we have main method in each class of same file.
    a. Yes
4. Main method variations

Keywords

1. Reserved words whose meaning is predefined in compiler
2. Example:
    a. **class**, **public**, **static**, **void**, **int**, **for**, **do**, **while**, **import**, **package** etc…

Identifiers

1. User[Programmer] defined words in program.
2. Name of the variable
    a. int **x**=45;
3. Name of the class
    a. class **First** { }
4. Name of the method
    a. void **m1**() { }
5. Recommended to use **meaningful** names
    a. Don't use **x** use **sum** for storing result.

Rule for Identifier

1. Combination of **letters(A-Z,a-z)**, **digits(0-1)**, **underscores(_)**, and **dollar($)** signs
   But **cannot** start from digit.
2. **Keywords** can not be used as Identifiers
3. Class names can be used as identifiers
4. Names are case sensitive

Variables

1. A variable is a **container** which holds the value while the Java program is executed
2. Variable is a name of memory location
3. int **x** = 50

Local Variables

1. Declared inside localscope
2. Inside method
3. Inside block like if,while,dowhile etc
4. It can not used outside that block scope

Instance Variable

1. Declared inside class and outside the methods
2. Used with object of class inside which they are declared
3. Copy of variables are made for each object
4. Syntax:
    a. object.Variable

Static Variables

1. Variable with static keywords are static variable
2. Single copy is made for static variable and shared amongst objects
3. Used directly with **className** no object is required
4. Syntax
    a. ClassName.VariableName

Data Types

1. Data Types **represent** a **type** of value that variable can hold.
2. **int** n1=50;
3. **double** n2=60.34;
4. **String** name=”Learn Code With Durgesh”;

Types of data type in java

1. Primitive Data types
2. Non-Primitive / User Defined Data Type / Reference Data types

Primitive Types

1. Inbuilt with programming language
2. There are **8 primitive data types** in java.
3. **byte**, **short**, **char**, **int**, **float**, **long**, **double**, **boolean**

Literals

1. Representation of fixed values
    a. Integer Literal
    b. Floating Point lateral
    c. Character Literal
    d. String Literal

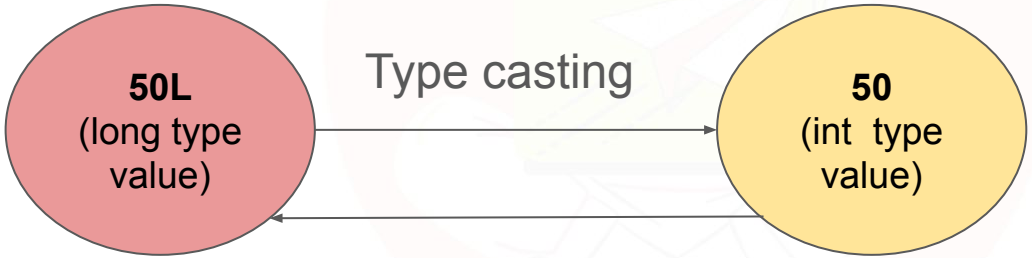
Non Primitive Data Types

1. Reference Data Type / User defined Data type
2. User defined data types is called non primitive data types
3. User defined as per need
4. Creating Types:
    a. We can use class to create types
    b. Class Student
    c. {

        }

Type Casting / Type Conversion

1. Converting one type value to another type.
2. Assign a value of one data type to another type.
3. Automatically
4. Manual



Implicit / Widening Casting

1. Automatic conversion of types
2. Mostly smaller to large
3. No change of loss of data

byte -> short -> char -> int -> long -> float -> double

Explicit / Narrowing Casting

1. Manual conversion of types
2. large to small types
3. Loss of data is possible
    a. double d=45.56
    b. float f=**(float)** d;

double -> float -> long -> int -> char -> short -> byte

Naming Conventions

1. Package name - **lowercase**
    a. java.lang
    b. java.util
    c. com.lcwd.controllers
2. ClassName - Pascal Convention
    a. MyProgram
    b. InetAddress
    c. InputStreamReader
3. Method name- Camelcase
    a. getName()
    b. doMyProgram()

Naming Convention

4. Variable Name- camelcase
    a. Meaningful name
    b. Should not use $ and _
    c. Don't use shortcut like n1,n2,m3
5. Constant : uppercase separated by _
    a. USER_ID
    b. CLIENT_ID
    c. CLIENT_SECREATE

Control Statements

1. Control Statements
    a. Decision Making
        i. If
        ii. Switch
    b. Looping Statements
        i. For
        ii. While
        iii. Do while
        iv. Enhanced for loop

Control Statements

1. Decision Making
    a. Jump statements
        i. Break
        ii. Continue

Practice Programs **on if..else**

1. WAP to find out number is even or odd
2. WAP to find number is divisible by 7 or not

Practice Programs on **if..else**

3. WAP to find maximum between two numbers.
4. WAP to check whether a number is negative, positive or zero.
5. WAP to check whether a number is divisible by 5 and 11 or not.
6. WAP to check whether a year is leap year or not.

Practice Programs on **ladder if/switch**

7. WAP to input marks of five subjects Physics, Chemistry, Biology, Mathematics and Computer. Calculate percentage and grade according to following:

Percentage >= 90% : Grade A

Percentage >= 80% : Grade B

Percentage >= 70% : Grade C

Percentage >= 60% : Grade D

Percentage >= 40% : Grade E

Percentage < 40% : Grade F

Practice Programs on **ladder if/switch**

8.WAP to input basic salary of an employee and calculate its Gross salary according to following:

Basic Salary <= 10000 : HRA = 20%, DA = 80%

Basic Salary <= 20000 : HRA = 25%, DA = 90%

Basic Salary > 20000 : HRA = 30%, DA = 95%

Practice Programs on **ladder if/switch**

9.WAP to input electricity unit charges and calculate total electricity bill according to the given condition:

For first 50 units Rs. 0.50/unit

For next 100 units Rs. 0.75/unit

For next 100 units Rs. 1.20/unit

For unit above 250 Rs. 1.50/unit

An additional surcharge of 20% is added to the bill

Switch statement

1. The switch statement can have a number of possible execution path.
2. It is like an if-else-if ladder statement
3. A switch works with the **byte**, **short**, **char**, and **int** primitive data types.
4. It also works with **enumerated types**, the **String** class, and a few wrapper classes: **Character**, **Byte**, **Short**, and **Integer**

Switch statement: syntax

```
switch(expression)
{
case value1:
 //code to be executed;
 break;  //optional
case value2:
 //code to be executed;
 break;  //optional
.....
 default:
  code to be executed if all cases are
not matched;
}
```

Points to remember about switch

1. There can be any number of cases but duplicates cases are not allowed.
2. The value for a case must be of the same data type as the variable in the switch.
3. The value for a case must be constant or literal. Variables are not allowed.
4. The break statement is used inside the switch to terminate a statement sequence.
5. The break statement is optional. If omitted, execution will continue on into the next case.
6. The default statement is optional .

Loops

1. Used to repeat the statements **n** number of times.
    a. For loops
    b. While loop
    c. Do_While loop
    d. Enhance For / For Each loop

Jump Statements

1. Break
2. Continue

Arrays

1. An array is a collection of similar type of elements which has contiguous memory location.
2. Collection of same type of elements referred by single variable
3. **Size is fixed**
4. **Homogeneous Elements**
5. **Zero-Based Indexing**
6. **Length Property**: length property provides the length
7. **Bounds Checking**
8. Arrays in are Objects: inherit the Object class

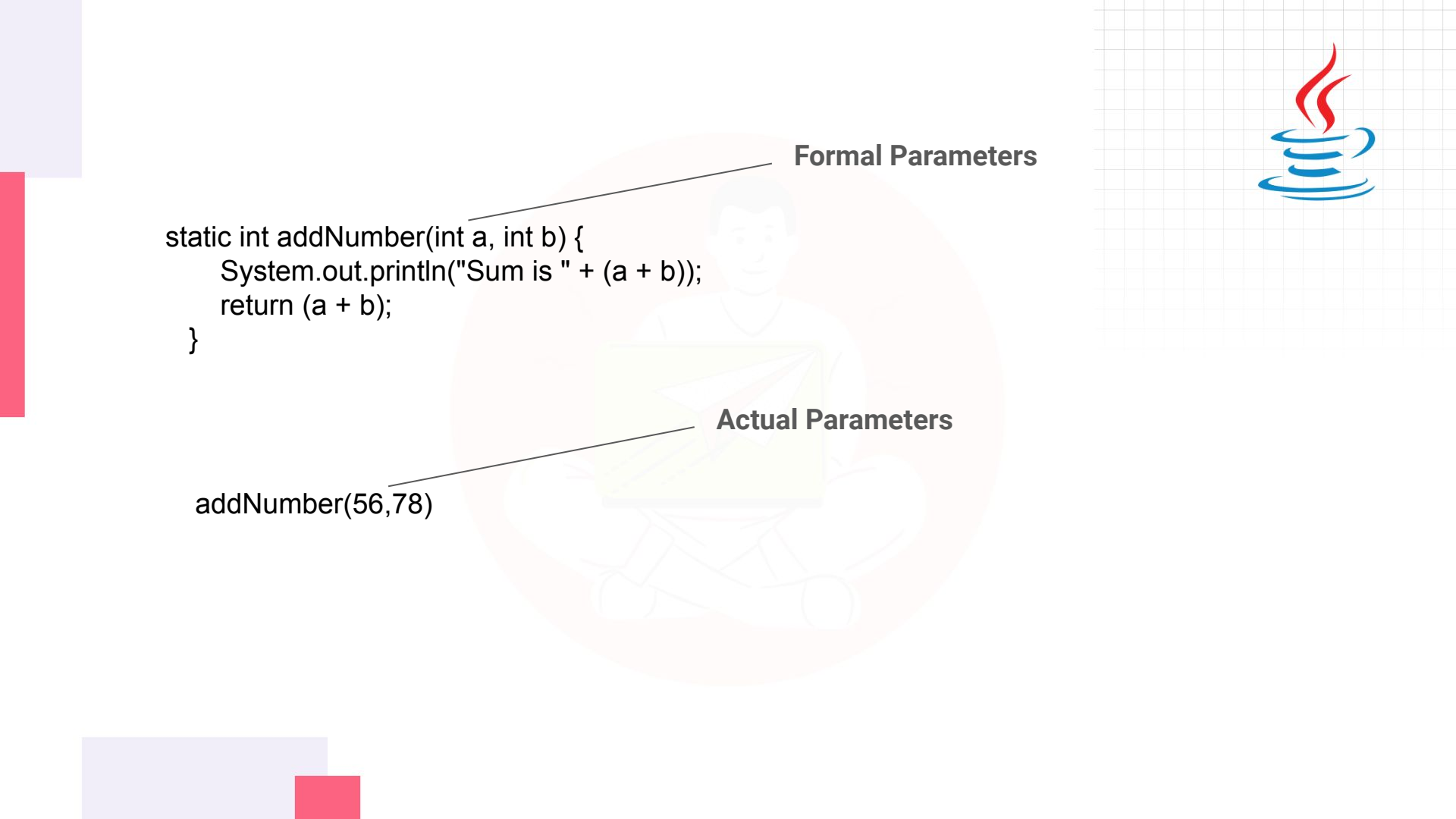
Array Types

1. 1D Array: collection of same type of data arranged in linear sequence.
2. 2D Array: A 2D array is an array of 1D arrays. It can be thought of as a table or grid of elements.

Array Practice Programs

Functions in Java

1. Set of statement written for doing specific task .
    a. Has name
    b. Accept values (Parameters)
    c. Return values
2. A method is a block of code which only runs when it is called.
3. Code reusability : Write code once call many times
4. Functions in java is basically a **methods**


Formal Parameters
static int addNumber(int a, int b) {
System.out.println("Sum is " + (a + b));
return (a + b);
}
Actual Parameters
addNumber(56,78)

Parameters

1. **Actual Parameters**: The actual parameter is the one that we pass to a function when we invoke it.
2. **Format Parameters** : The formal parameter is one that we pass to a function when we declare and define it.

Call by Value: Calling method with value

1. Value of actual parameter is **copied** to formal parameter.
2. Any changes made in parameter inside method is not reflected to actual value.

Call by Reference: Calling method with Reference (Object)

1. Value of actual parameter is **transferred** to formal parameter.
2. Any changes made in parameter inside method is reflected to actual value.

Practice All Given Programs

Java OOPs Concepts

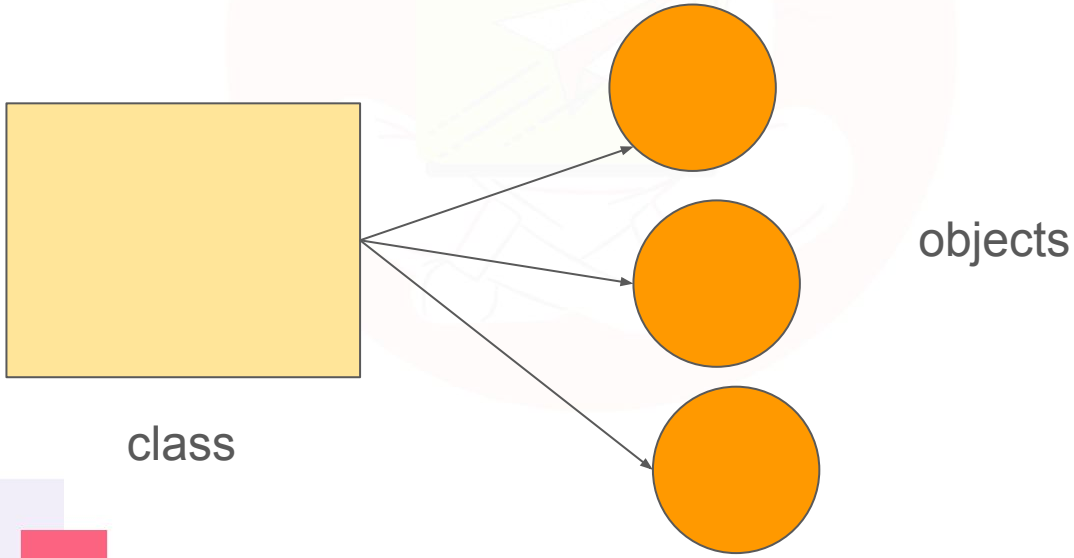
1. **OOPs:** Object Oriented Programming System
2. Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects.
3. Object Oriented Provides Concepts likes
    a. Object
    b. Class
    c. Inheritance
    d. Polymorphism
    e. Abstraction
    f. Encapsulation

Objects and Classes

1. **Object:**
    a. An entity that has **states**(attributes/properties) and **behavior** is known as an object like car, chair, pen etc
    b. Real world entity.
    c. Objects also has unique identity.
    d. Dogs have state (**name, color, breed, hungry**) and behavior (**barking**, **fetching**, **wagging** tail).

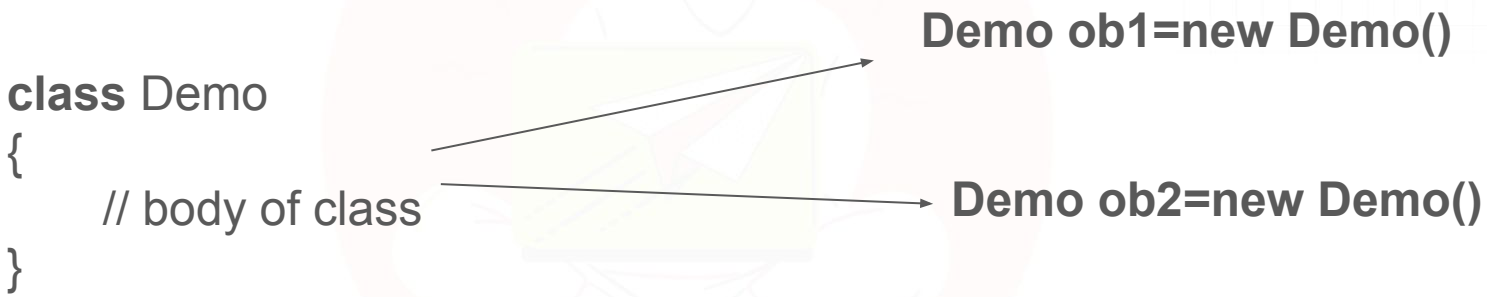
Objects and Classes

1. **Class:**
    a. Blueprint from which individual objects are created.
    b. Logical Entity.
    c. Object is instance of class.
    d. Class can have **properties** and **behaviours** declared .



Creating Class and Objects

1. Class can be created with **class** keyword.
2. Object is created with **new** keyword.



Variables and Methods in Classes

1. Classes are only blueprint for objects
2. Variables are called **data members/properties. (**basically separate copy are created for each object**)[talking about instance variable here]**
3. Methods are called member **methods/behaviors/functionality.(**each methods get executed in object spaces).[talking about instance/non-static methods]
4. Methods: set of instruction return for doing specific task.(jo ham pahle padh chuke hai.)
5. Lets try with code.

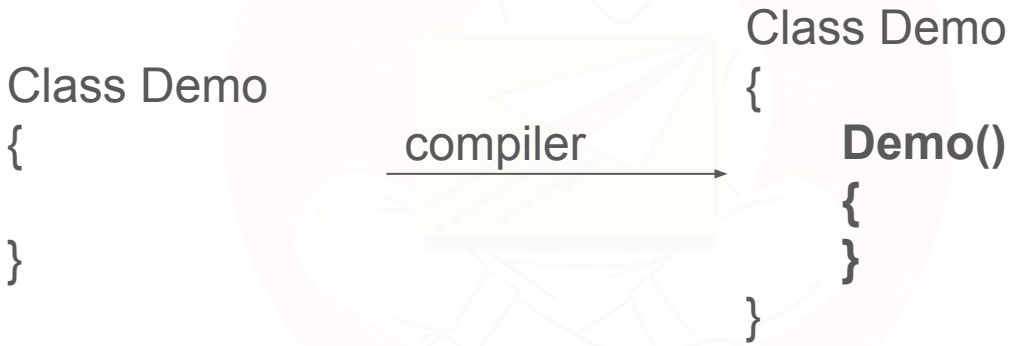
Constructor

1. Block of code gets executed automatically when object is created.
2. Similar to methods.
3. Constructor and **class name** must be same.
4. Constructor **does not** return any value.
5. Example.
6. Type of constructor
    a. Parameterized - constructor that takes 1 or more parameters in argument list.
        i. new Demo(int a, int b);
    b. Non-Parameterized- constructor that does not take any parameters in argument list
        i. new Demo();

.

Constructor

1. It is not important to create any constructor in java class. Because when we don't create any constructor in class then java compiler create on default constructor for us.



Overloading

1. When we defined more than **one constructor** or more than **one methods** with same name then it is called **Overloading**.
2. Constructor overloading:
    a. When we defined more than one constructor in class .
        i. Argument must be different for overloading
        ii. Number of parameters / type of parameters / order of parameters
        iii. At Least one of above must be different for overloading.
3. Method Overloading:
    a. When we defined more than one methods with same name but of different argument list.

Conditions for method overloading

1. Argument must be different for overloading
2. Number of parameters / type of parameters / order of parameters
3. At Least one of above must be different for overloading.
4. Name of the method must be same.
5. Return type can be different.
6. Methods must be in same class.😂

‘this’ keyword

1. **this** keyword is used to remove the naming conflict between local and instance variable of same name.
2. **this** use to call current class methods and variables.
3. **this()** use to call current class constructor from inside another constructor.
4. this keyword refer the current invoking object.
5. Passing current object.
6. Method chaining.

‘static’ keyword

1. To create static variables.
    a. **static int a=50;**
2. To create static methods.
    a. public **static** void test() {
    }
3. To create static blocks.
    a. **static**{
    }

‘static’ execution flow

1. Class Loading:
2. Static Variable Initialization:
    a. Default values
    b. Explicit initialization
    c. Static Block Execution (this is where they fit in)
3. Static Method(MAIN) Execution:

POINTS

1. Static blocks are only executed once, when the class is first loaded.
2. They cannot access instance variables or instance methods, as those only exist within objects.
3. They can access other static variables and methods of the class.

Inheritance (IS-A Relationship)

1. It's a mechanism where a **new class** acquires the **properties** and **behaviors** of an existing class.
2. Class which provides the properties and behaviors is called **Parent Class/ Super Class/ Base Class.**
3. New class which takes the properties and behaviours is called **Child Class / Sub Class/ Derived Class**.
4. Code Reusability: Subclass reuses code from the superclass, avoiding duplication.
5. Extension: Subclass can add new properties and methods to extend functionality.
6. **extends** keyword is used to do inheritance.

Types of Inheritance:

1. **Single Inheritance:** A subclass inherits from only one superclass.
2. **Multilevel Inheritance**: A subclass inherits from a superclass, which itself inherits from another superclass.
3. **Hierarchical Inheritance:** Multiple subclasses inherit from a single superclass.
4. **Multiple Inheritance:** A subclass inherits from multiple superclasses (not directly supported in Java, but achieved through interfaces).
5. **Hybrid Inheritance** : Involving more than one inheritance.
6. **Cyclic Inheritance** : When cycle is form in hierarchy.

Inheritance is a powerful tool for building well-structured, reusable, and extensible Java applications.

‘super’ keyword

1. Refer the immediate parent class object.
2. Accessing Superclass Members:
3. Calling Superclass Constructors:[**super()**]

Points to remember

1. Use this to refer to the current object's members.
2. Use super to refer to the superclass's members.
3. super must be the first statement in a subclass constructor if you're calling a superclass constructor.
4. super cannot be used in static methods or blocks, as they don't belong to specific objects.

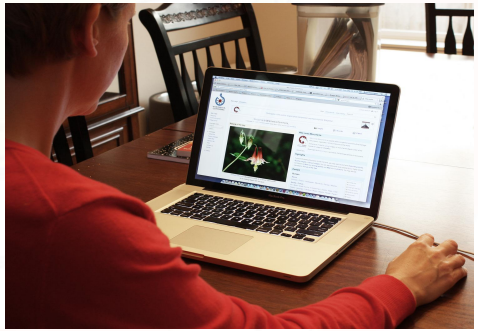
Overriding

1. Redefining the parent class methods in child class is called overriding.
2. The overriding method must have the **same signature** (**name**, **return type**, and **parameter list**) as the superclass method.
3. The overriding method can have a **broader access** modifier (e.g., public overriding protected).
4. The overriding method cannot have a narrower return type than the superclass method.
5. The overriding method can throw any unchecked exceptions, but it cannot throw new checked exceptions or narrower checked exceptions than the superclass method.
6. Constructors cannot be overridden.

Abstraction

1. Abstraction in Java is the process of hiding implementation details and exposing only essential features to the user.
2. It focuses on what an object does rather than how it does it.
3. Examples:

Driving Car



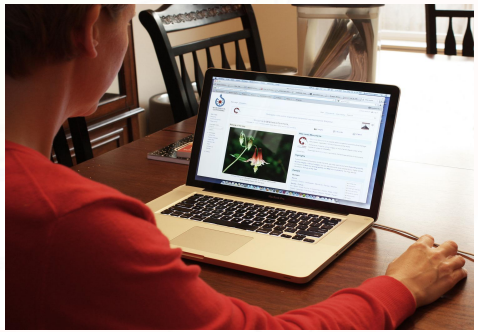
Using Computer

ATM Machine

How to achieve Abstraction

1. Using Abstract Classes
2. Using Interfaces

Driving Car



Using Computer

ATM Machine

Abstract Classes

1. Abstract classes are the incomplete classes.
2. The class that contain at least one abstract method is abstract class.
3. Abstract classes are classes that cannot be instantiated directly.
4. They serve as blueprints for subclasses to inherit and implement.
5. They are declared using the **abstract** keyword.
6. They can contain both abstract methods (without implementation) and concrete methods (with implementation).
7. Abstract classes provide **0% to 100%** abstraction level.

Interfaces

1. Interfaces are similar to abstract classes but interfaces provide 100% abstraction.
2. An interface defines a contract for a set of methods that a class must implement.
3. In interfaces all variables are by default **public static final**.
4. All methods automatically becomes **public abstract**.
5. **interface** keyword used to create interface.
6. **implements** keywords is used to implement interfaces.

Important Point

1. Interface methods cannot have access modifiers (all public).
2. Interface variables by default **public static final.**
3. Default methods and static methods can be added to interfaces in Java 8 and later.
4. Marker interfaces define no methods but serve as flags or tags for class behavior.

Polymorphism

1. Ability of Objects to behave differently
2. Polymorphism as the ability of a message to be displayed in more than one form.
3. Man at the same time is a father, a husband, and an employee. So the same person possesses different behaviors in different situations
4. The word **“poly”** means **many** and **“morphs”** means **forms**, So it means many forms.
5. Type:
    a. Compile-Time Polymorphism (Static Polymorphism):
    b. Runtime Polymorphism (Dynamic Polymorphism):

Compile Time (Static Polymorphism)

1. Resolved at **compile time**.
2. Achieved through **method overloading**.
3. The compiler selects the appropriate method based on the arguments passed at compile time.

RunTime (Dynamic Polymorphism)

1. Resolved at **run time**.
2. Achieved through **method overriding**.
3. The actual method to be invoked is determined at runtime based on the object's type.

Encapsulation

1. Wrapping up of data member(variables) and member methods (method) in single unit is called ecasulation.
2. Provide Security
3. Modular Approach
4. Maintenance is easy

final keywords

1. Variables: final variable cannot be changed once initialize.
2. Class: final class cannot be inherited .
3. Methods: final methods cannot be overrided

Instanceof operator

1. The java instanceof operator is used to test whether the object is an instance of the specified type (class or subclass or interface).
2. The instanceof in java is also known as type comparison operator because it compares the instance with type. It returns either true or false. If we apply the instanceof operator with any variable that has null value, it returns false.

Concept of Packages

1. In Java, packages are like folders that organize classes and interfaces into related groups.
2. Packages are a way to group related classes and interfaces together based on functionality, domain, or purpose.
3. Think of them as folders organizing files on your computer; packages organize your code for better clarity and maintenance.
4. Each package has a unique name, typically following a reverse domain name format (e.g., com.example.myapp).

Advantages of Packages

1. Remove Naming Conflication
2. Improved code organization
3. Access control
4. Modularization
5. Easy to maintain , understand , modify the code.

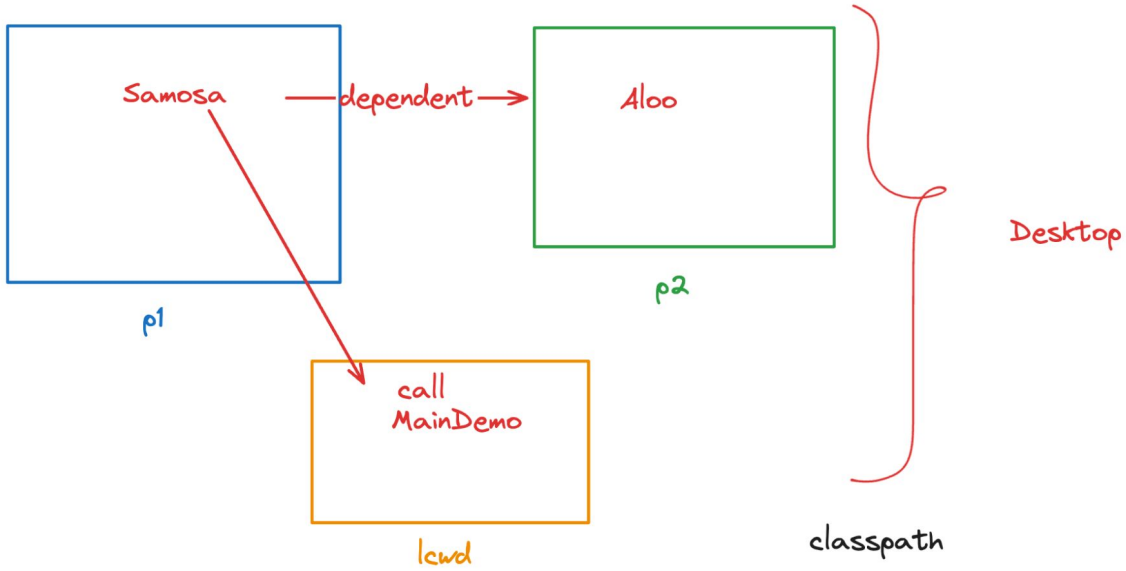
Type of packages in java

1. Predefined packages: packages that java provides to us
    a. java.util
    b. java.lang
    c. Java.net etc
2. User defined packages : packages that we create .

How to create packages manually

1. create folder with package name. same for subpackage
2. create java file within same folder.
3. declare package with package keyword

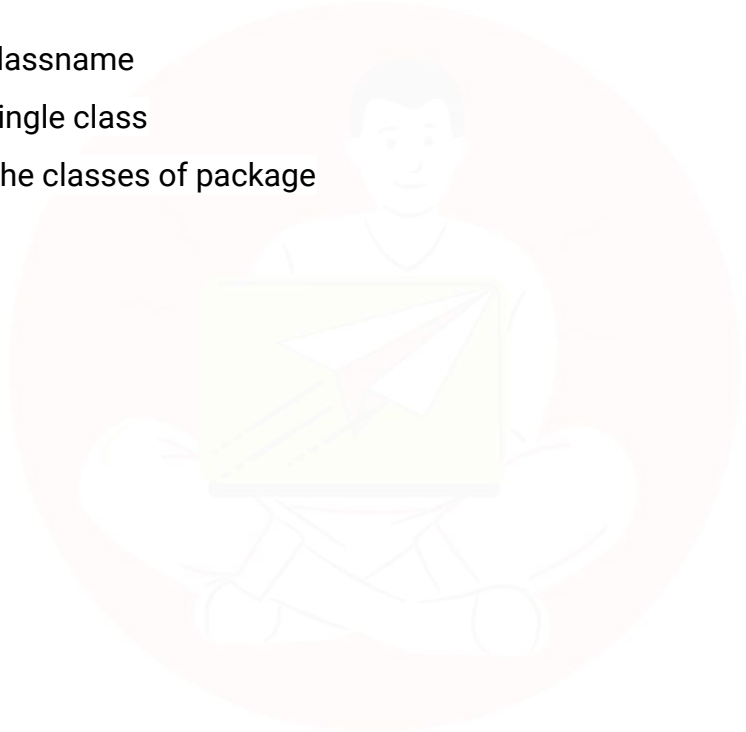
Creating Complex use case:



javac -cp ./:./durgesh/tiwari lcwd/DemoMain.java

Accessing classes of package

1. use: fully qualified classname
2. import: import the single class
3. import *; import all the classes of package



Access Modifiers

1. Access modifiers controls the accessibility of a class, constructor, variable, method, or data member.
2. public
3. private
4. default
5. Protected

public>protected>default>private

Access Modifiers

modifiers	same class	same package subclass	same package non subclass	different package subclass	different package non subclass
private	✓	✗	✗	✗	✗
default	✓	✓	✓	✗	✗
protected	✓	✓	✓	✓	✗
public	✓	✓	✓	✓	✓

JAR(Java Archive Files) files

1. In Java, a JAR (Java Archive) file is a compressed file format that bundles together multiple Java class files, associated metadata, and resources (such as images, text files, etc.) into a single archive
2. The primary purpose of JAR files is to facilitate the distribution and deployment of Java applications or libraries.

JAR Basic operations

1. Creating jar file
    a. jar -cvf filename.jar *
2. Extracting jar file
    a. jar -xvf filename.jar
3. Creating jar with main file
    a. jar -cvfe filename.jar mainClass *
4. Executing jar file
    a. java -jar filename.jar

String Handling

1. Set of character placed within double quotes is called string.
    a. “Abc”
    b. “235256”
    c. “This is great book”
2. Performing operations on string is called string handling.

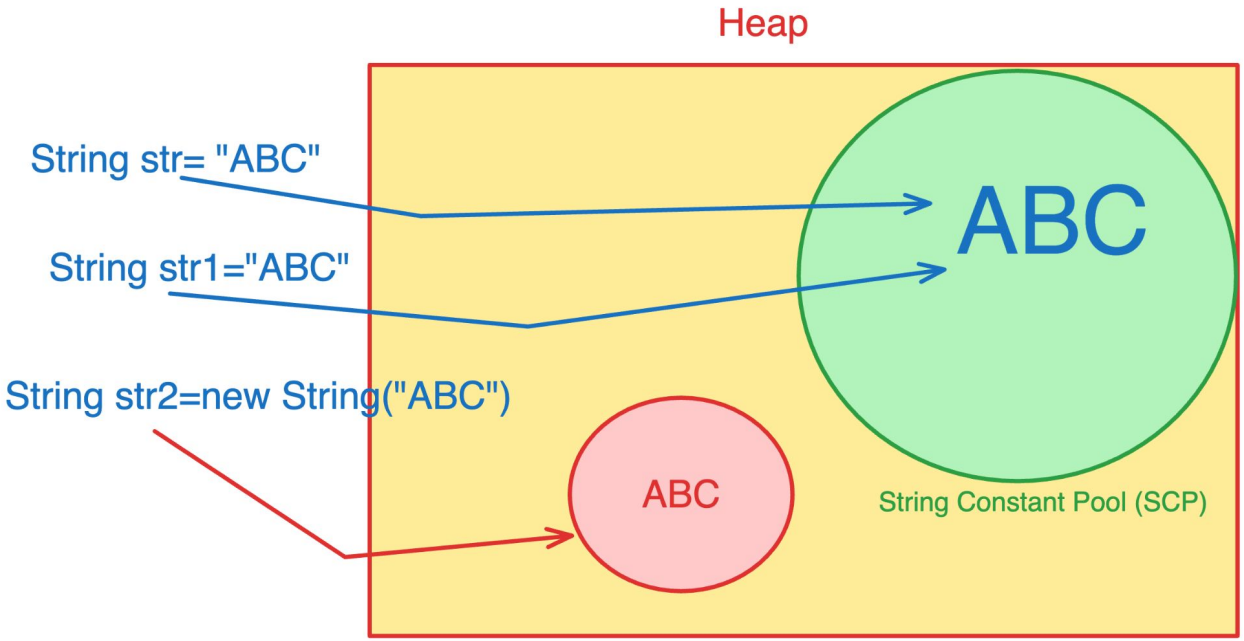
Important Classes

1. java.lang.String
2. java.lang.StringBuffer
3. java.lang.StringBuilder
4.

String Pool Constant / String Constant Pool

1. It's a special memory area within the heap that stores unique, immutable string literals created during program execution.
2. When you create a string literal using double quotes (e.g., "Hello World"), the JVM checks if an identical string already exists in the pool.
3. If it exists, the existing string reference is returned instead of creating a new object.
4. This effectively creates a single instance of the string, saving memory and improving performance.

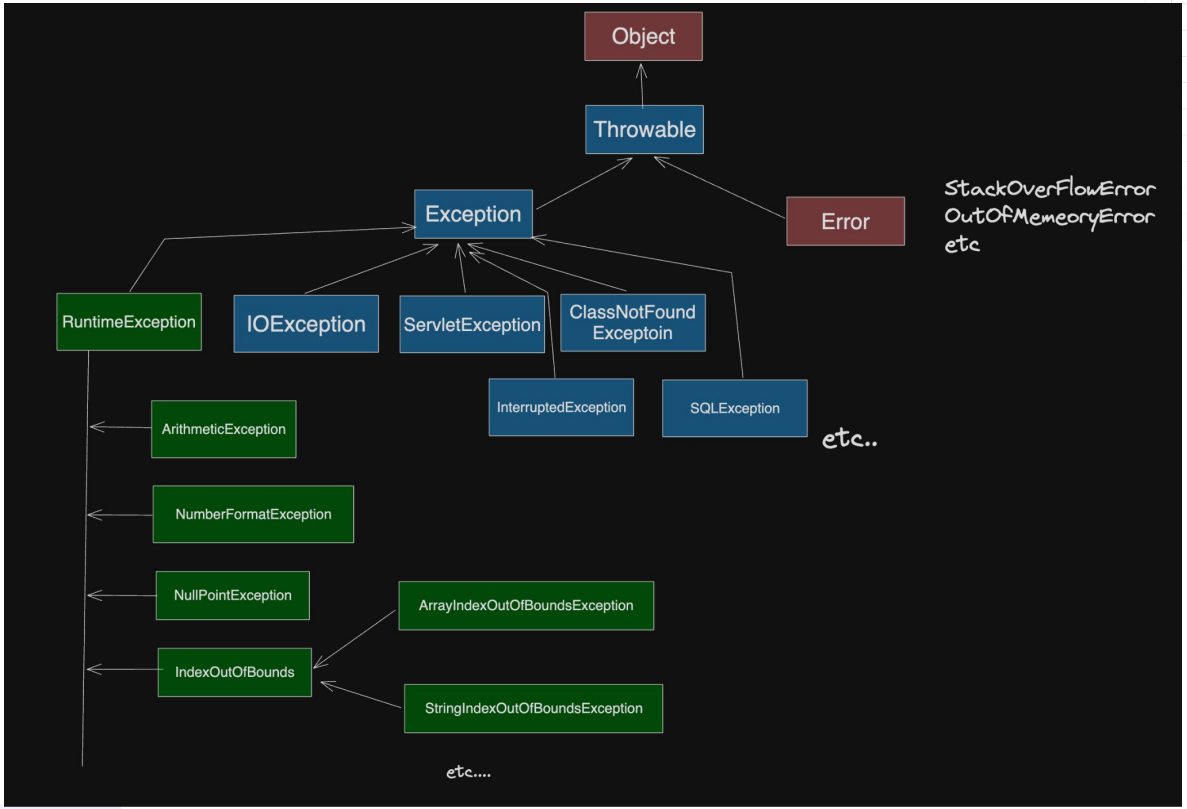
String Pool Constant / String Constant Pool



Exception Handling

1. **Exception**: An unwanted event that **disrupt** normal flow of execution of our program.
2. **Exception Handling**: Mechanism of handling exception to maintain normal flow of execution of program.
3. To maintain the normal flow of execution of program we need to handle exception in programs .

Exception Hierarchy

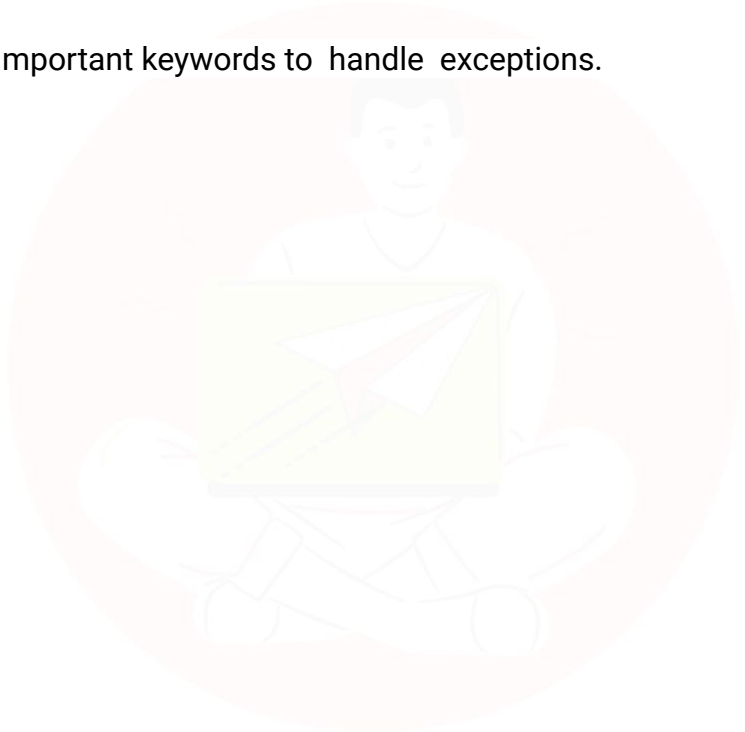


Type Of Exception

1. **Checked Exception**: The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions.
2. **UnChecked Exception**: The classes that inherit the RuntimeException are known as unchecked exceptions.
3. **Error**: Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

How to handle exception in java ?

- Java provides 5 important keywords to handle exceptions.



1. try
2. catch
3. finally
4. throw
5. throws

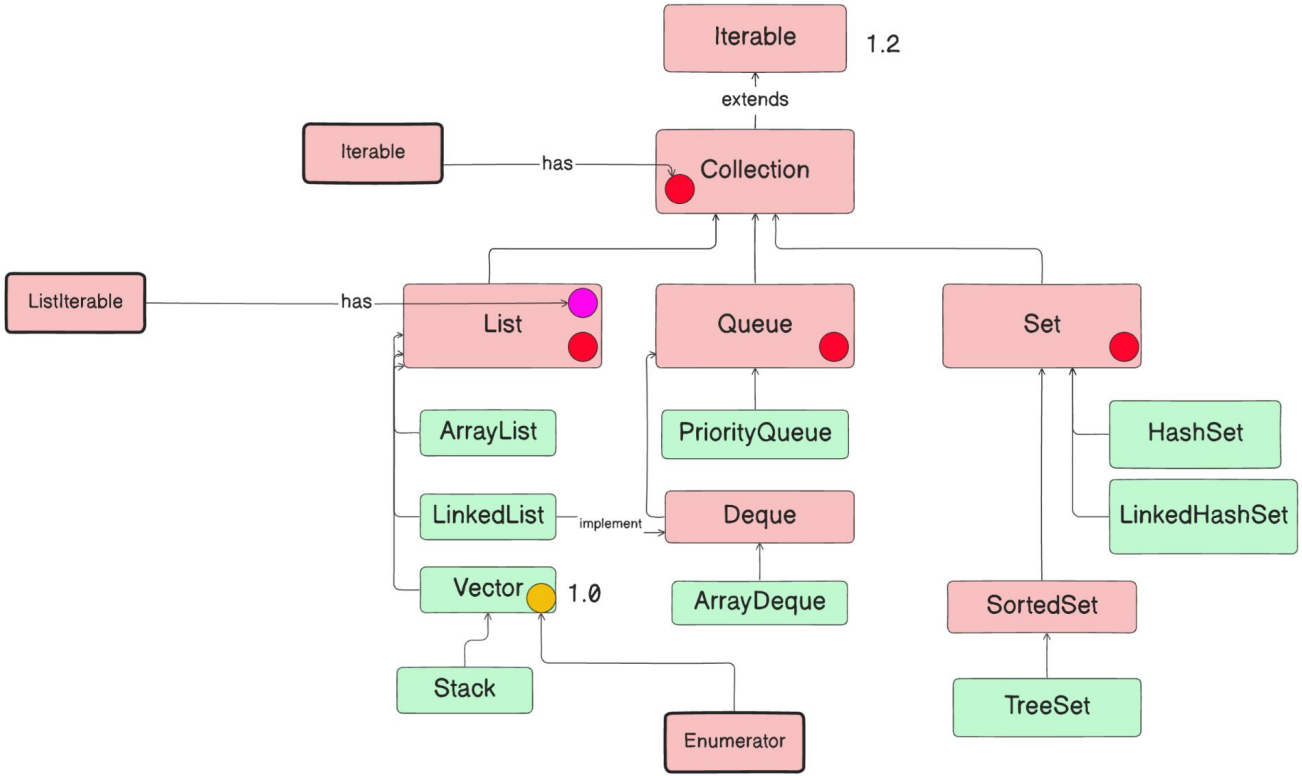
Collection Framework

1. **Collection**: A collection is an object that represents a group of objects.
2. **Collection Framework :** The Collection framework represents a unified architecture for storing and manipulating a group of objects.
3. Collection Framework has :
    a. Set of classes and interfaces (Interfaces and its implementations)
    b. Algorithms
4. java.util package provides all the classes and interfaces of collection framework

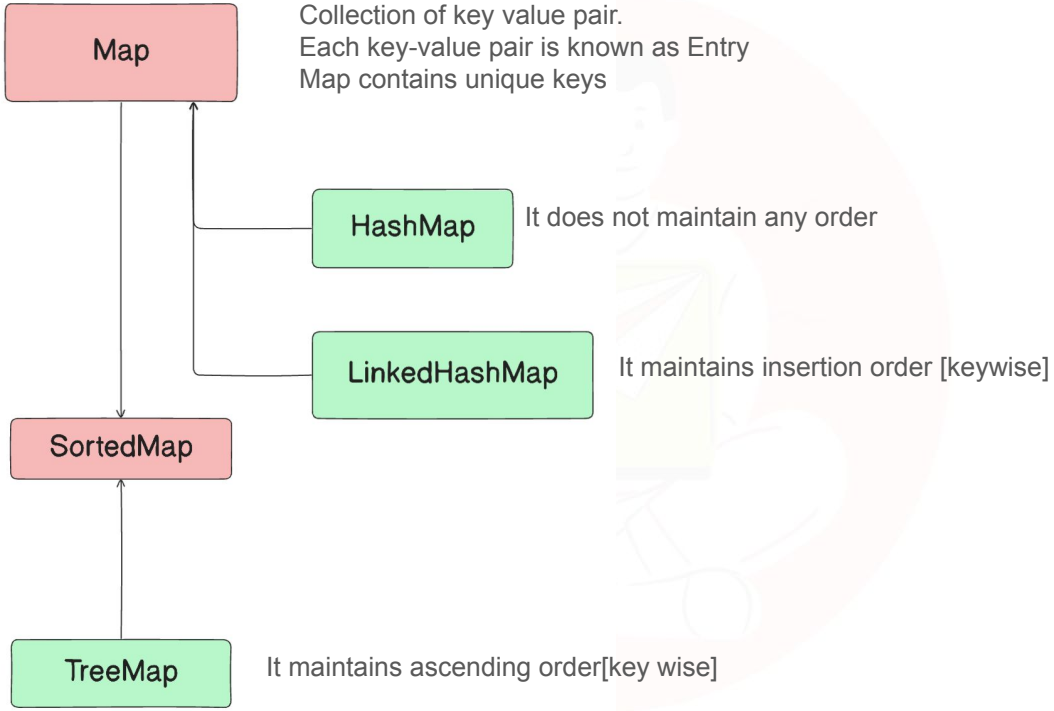
Benefits

1. Reduces programming effort by providing ready-made classes and interfaces.
2. Enhances performance with optimized implementations.
3. Promotes code reusability and consistency.

Collection Hierarchy(Collection)



Collection Hierarchy(Map)



Implementation Class Summary

Interface	Hash Table	Resizable Array	Balanced Tree	Linked List	Hash Table + Linked List
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Deque		ArrayDeque		LinkedList	
Map	HashMap		TreeMap		LinkedHashMap