1. Class – Blueprint to create objects

* It consist of member variables (state) & member methods (behavior)
* Class is the combination of state & behavior
* Types of Class
  + **Starter Class** – A class with a main method (Only this type of class can be directly run in JVM as java application)
  + **Non-Starter Class** – A Class without main method
  + **Concrete/Complete/Simple Class** – A class without any abstract methods
  + **POJO** – Plain Old Java Object – Class which is not extending any other class nor implementing any interface
  + **Abstract Class/In-complete/Non-concrete** – A class with one or more abstract methods (abstract methods will not have body only declaration no definition)
  + **Bean Class** – A class with few properties, constructor and getters & setters
  + **Entity Bean Class** – A Bean class which represents a table in the database
  + **Base/Parent/Super Class** – A class which is providing it’s public members to other classes
  + **Derived/Child/Sub Class** – A Class who is receiving the public members of other class
  + **Wrapper Classes** – These classes are used to convert primitive to it’s corresponding object [Boolean, Byte, Short, Character, Integer, Float, Long, Double]

1. Objects – Instance of a class

* Each object will have it’s own value for the properties

class Employee {

int id;

String name;

}

main() {

Employee emp = new Employee();

emp.id =100;

emp.name =”Test”;

Employee emp1 = new Employee();

Emp1.id =101;

Emp1.name =”ABC”;

}

* Objects will be created in heap area when using new keyword along with the Constructor of the class
* During object creation time, the memory will be allocated for the objects.

Employee emp; --- Creating an Employee reference variable. Here object will not be created and memory also will be allocated for object. (Reference variable is a kind of pointer)

* Reference variables usually stores the address of objects.
* <https://techvidvan.com/tutorials/java-virtual-machine/> JVM Architecture

1. Access Modifiers

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sl No | private | package/default | protected | public |
| Within the same class | Yes | Yes | Yes | Yes |
| Within the same package | No | Yes | Yes | Yes |
| SubClass in different package | No | No | Yes | Yes |
| Anywhere | No | No | No | Yes |

* Access modifiers can be used in classes, interfaces, methods, enums and also properties

1. Non-Access Modifiers (Behavior Modifiers)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| final | abstract | static | transient | volatile | strictfp |
| Make the state constant | --- | A single copy of variable shared by all the objects | Is used to avoid serializing a particular property |  |  |
| Makes the method non-overridable | Used to declare abstract /in-complete methods | Static methods can be accessed using the class name | --- |  |  |
| Makes the class non-extendable | Used to declare abstract/in-complete class | --- | --- |  |  |

OOPs – Object Oriented Programming – Object is the main entity, everything depends on the object

OOP – Is a General Programming Concept.

There are Object Oriented Programming Languages (C++, C#, Java, Python …..)

N.B : Java is not a pure Object Oriented Programming Language, because it contains primitive data type which are non-objects.

A PIE – Abstraction Polymorphism Inheritance Encapsulation

|  |  |  |  |
| --- | --- | --- | --- |
| OOP |  |  |  |
| Abstraction | Hiding the implementation | By Using abstract classes & Methods | By Using Interfaces |
| Polymorphism | Reusing the same method name for different operations | Compile time / static polymorphism (method overloading) | Runtime/ Dynamic Polymorphism (method overriding) |
| Inheritance | Sharing the state and behavior between two or more classes | extends keyword for classes | implements keyword for interfaces |
| Encapsulation | Securing the Data (Hiding the data) by capsulating member variables inside the class using private access modifier | By making the state private and making the behavior public |  |

Casting – Type Casting – Refers to converting one primitive format to another primitive format. (primitive 🡪 primitive) (Object 🡪 Object) [Object 🡪 String]

Types

* Explicit Type Casting - Narrowing – Bigger to Smaller value – Need to specify the target data type in side the paranthesis. (Manual Casting)
* Implicit Type Casting – Widening – Smaller to Bigger value – No need to use any keywords – Automatically done by the JVM (auto-casting)

Boxing – Conversion of primitive data type to it’s corresponding object representation with the help of wrapper classes.

(primitive 🡪 Object)

* Auto-Boxing – Usually it happens when we supply primitive data to collections.
* Auto -Unboxing – Reverse of boxing

String – represents a String Object – Which is sequence of characters

String is special data type in JAVA. String represents immutable objects meaning it can’t be modified after creation.

String is the only type in java which supports primitive declaration [String str=”Hello”] & Object declaration [String strObj = new String(“Welcome”);]

String class is defined in java.lang package (Is the default package in Java which gets imported automatically by the JVM when you create a class)

String is not a primitive data type. It represents object only.

Object Class – Object is the root class for all user-defined classes. This is also defined in java.lang package

All Wrapper classes are defined in the java.lang package

Official Java SE 8 documentation --- <https://docs.oracle.com/javase/8/docs/api/>

Latest version of JAVA – JAVA 17.

We are using Java 1.8 (Java 8)

int a =5; // primitive declaration

int rollNumbers[] = new int[25]; // primitive int array

Data types

* Primitive [boolean, byte, short, char, int, float, double]
* Non-Primitive (Derived) [ arrays, enums, object references]

Abstract Classes

* Abstract Class can’t be instantiated (Objects can’t be created) directly.
* Objects of abstract class can be created using anonymous inner class (In-direct)
* If a class contains an abstract method then the class is an abstract class
* Abstract class is also called as non-concrete/ in-complete class
* Abstract class contains zero or more in-complete/non-concrete method
* Abstract classes can be extended to create sub classes

Difference between abstract class and interface – when to use abstract class & when to use interface

|  |  |  |
| --- | --- | --- |
| Sl No | Abstract Class | Interface |
| 1 | It will have both concrete and abstract methods | It will have concrete & abstract methods |
| 2 | By default all the methods are non-abstract. Here abstract keyword is necessary to declare abstract methods | By default all the members are public & abstract only. So that, the abstract keyword is not necessary in interfaces. |
| 3 | It can also have static, default & final methods | It can have only static & default methods in addition to abstract method |
| 4 | It can have zero or more abstract method | It can have zero or more abstract method |
| 5 | A class can extend only one class | Many interfaces can be implemented by a class |

Interfaces – Acts as a contract – which defines the behavior (methods) which the class needs to implement (needs to provide functionality for zero or more methods)

All the member variables defined in interface are public, static & final by default.

All the member methods defined in interface are public & abstract unless specified otherwise.

Interface can also have some static & default concrete methods – Methods with body.

Abstract methods – method without body

Concrete Methods - method with body

Exception Handling –

Exception – Is an un-expected situation while running the program- which leads to pre-mature termination of the program.

When exception object is thrown JVM will catch it and immediately stops executing the program.

If handled properly using try/catch block, pre-mature termination can be avoided.

Exception Indicates – something went wrong when executing the program or when compiling the program

Exception Handling – Way to control and handle the exception

Two ways of Exception Handling

1. Using try/catch/finally block (Recommended)
2. Using throws keyword (Not-recommended – Passing the responsibility to the JVM) – ducking the exception

Types of Exception

1. Checked Exception /Compile time Exception – Without handling it, java code can’t be run
2. Un-Checked Exception / Runtime Exception – It’s not compulsory to handle it.

Exception Hierarchy

Throwable – Is a Top class in Exception hierarchy

* Errors
* Exceptions
  + Runtime Exception /Un-Checked Exceptions – Arithmetic, ArrayIndex, NullPointer,
  + Compile time Exception /Checked Exceptions – IOException, ClassNotFound, SQL
* IllegalStateException is associated with Thread class. When we try to call the start() method second time. IllegalThreadStateException is the sub class of IllegalStateException.
* Checked Exceptions needs to be handled before executing the code
* Checked exceptions will be thrown during the compile time, compiler will not allow you to run the code without handling the checked exceptions (compile-time Exceptions)

Types of try/catch block

* A try block can have multiple catch block
* A try block must be followed by a either a catch block or by a finally block
* A try block can have both catch block and finally block together.
* A try block with catch block alone is also valid
* A try block with finally block alone is also valid
* When we have multiple catch block following the try block, it is important to handle the child-class exception object first and then the parent class exception (Arithmetic, ArrayIndex, Exception. Otherwise code will throw un-reachable code error.
* A catch block can handle multiple exception objects [ catch(ArrayIndex | Arithmetic | SQL | FileNotFound e)]
* When System.exit() is called – this is the time, the finally block code can be skipped.
* Try with resource -This will declare a resource along with the try. This resource will be closed automatically while coming out of the try block.

Example try (File f = new File(“test.txt”)) {

} catch (Exception e) {

e.printStackTrace(); //It’s a costly process

}

* Finally block is used to do clean-up activities like closing the resource (file/db connection/stream/ scanner)
* The code inside the finally block will gets executed all the time irrespective of the exception status

Collection – Is the way of handling group of objects

* There are many classes and interface to work with group of objects
* List (I) – It allows duplicates, maintain insertion order, [ArrayList, LinkedList]
* Set – It won’t allow duplicates, will not maintain insertion order,
* Map – It stores the data in key,value pair.

Stack – Data Structure – Data Structure refers storing and handling the data in structured format.

Stack – LIFO – Last In First Out [Stacking coins]

Queue – FIFO – First In First Out [Ticket Booking Queue]

Collection will work with objects only not with primitive.

When we supply a primitive data to a collection, jvm will automatically convert it to objects. This is called auto-boxing.

Calculating interest for different types of account in a bank for a period of three months.

Utility classes available in Collection API

* Collections class – Using this class, we can search, sort or reverse find max ,min value of collection objects
* Comparable, Comparator
* Iterable, Iterator

**Collection** is a interface where as **Collections** is a class.

In Wrapper class parseXX() – method is used to convert Object to primitive, valueOf() method is used to convert primitive to object.

