# WEEK-1

1. **Advantage of OOPs**

- Easy Troubleshooting
- Reuse of code through inheritance
- Flexibility through Polymorphism
- Effective problem Solving (breakdown of software and then solve)

2. **Object and Class**

Object − Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors – wagging the tail, barking, eating. An object is an instance of a class.

Class − A class can be defined as a template/blueprint that describes the behavior/state that the object of its type supports.

3. **Method Overloading**

- done

**4.Constructor**

- done

**5.static variable, method and block**

- **Static variable -** when variable is declared with word static, it cannot be changed again
- **Static Method -** If you apply a static keyword with any method, it is known as static method.
● A static method belongs to the class rather than the object of a class.
● A static method can be invoked without the need for creating an instance of a class.
● A static method can access static data members and can change the value of it.
-
- **Static Block -**
● Is used to initialize the static data member.
● It is executed before the main method at the time of classloading.
-

**6.this keyword**

Here is given the 6 usage of java this keyword.

- this can be used to refer current class instance variable.
- this can be used to invoke current class method (implicitly)
- this() can be used to invoke current class constructor.
- this can be passed as an argument in the method call.
- this can be passed as argument in the constructor call.
- this can be used to return the current class instance from the method.

**7.Inheritance (IS-A)**

**8.Aggregation and Composition(HAS-A)**
**9.Method Overriding**
**10.Covariant Return Type**

**Covariant return types**

Java 5.0 onwards it is possible to have different return type for a overriding method in child class, but child's return type should be sub-type of parent's return type. Overriding method becomes variant with respect to return type.

Co-variant return type is based on **Liskov substitution principle.**

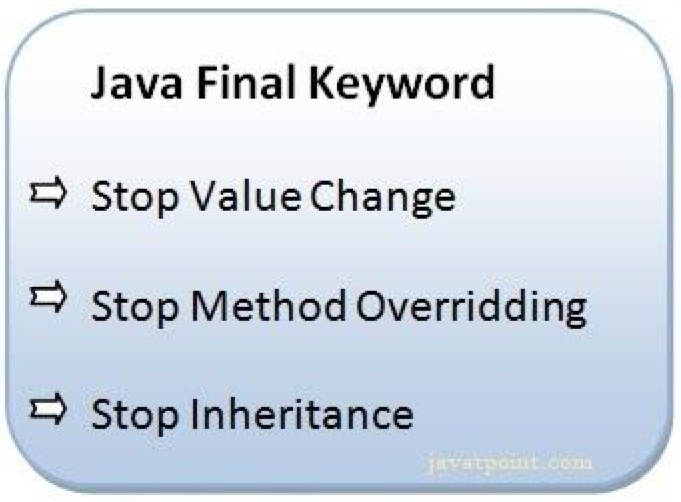
**11.super keyword**

- Invoke parent variable
- Invoke parent method
- Invoke parent constructor by super();

**12.Instance Initializer block**

- IIB is created when instance of the class is created
- Executed when class is created

**13.final keyword**


Java Final Keyword

- Stop Value Change
- Stop Method Overridding
- Stop Inheritance

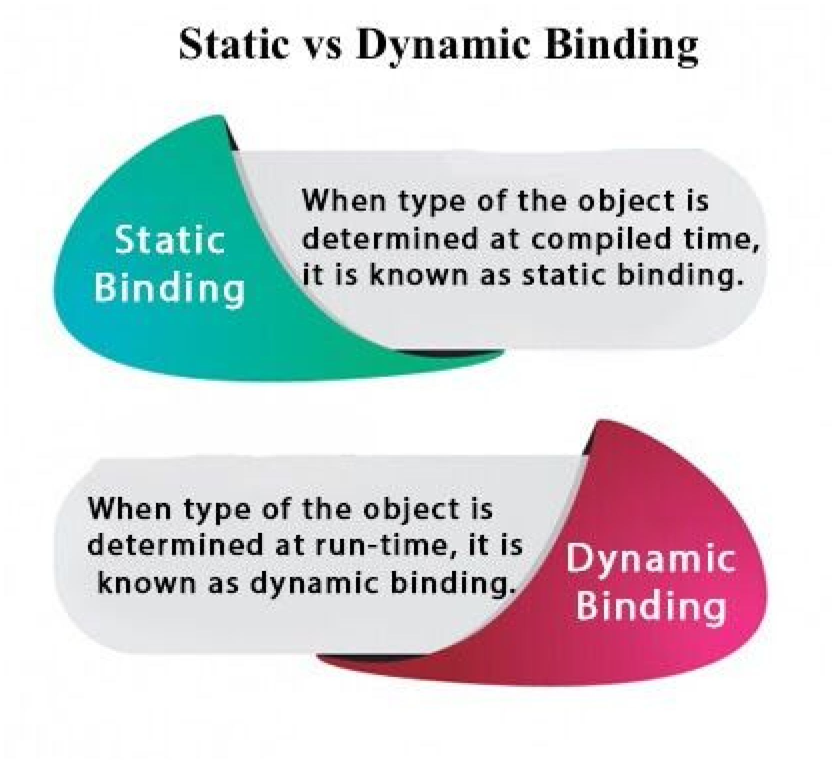
javatpoint.com

**14.Runtime Polymorphism**

Class A ()
Class B extends A()

A a = new B() - this overrides method in A with B

**15.static and Dynamic binding**


Static vs Dynamic Binding
Static Binding
When type of the object is determined at compiled time, it is known as static binding.
When type of the object is determined at run-time, it is known as dynamic binding.
Dynamic Binding

Class A ()
    Class B extends A()

    A a = new B() - this is dynamic binding

**16.Abstract class and Interface**

| Abstract class | Interface |
|---|---|
| **1) Abstract class can have abstract and non-abstract methods.** | **Interface can have only abstract methods. Since Java 8, it can have default and static methods also.** |
| **2) Abstract class doesn't support multiple inheritance.** | **Interface supports multiple inheritance.** |
| **3) Abstract class can have final, non-final, static and non-static variables.** | **Interface has only static and final variables.** |
| **4) Abstract class can provide the implementation of interface.** | **Interface can't provide the implementation of abstract class.** |

| | |
|---|---|
| **5) The abstract keyword is used to declare abstract class.** | **The interface keyword is used to declare interface.** |
| **6) An abstract class can extend another Java class and implement multiple Java interfaces.** | **An interface can extend another Java interface only.** |
| **7) An abstract class can be extended using keyword "extends".** | **An interface can be implemented using keyword "implements".** |
| **8) A Java abstract class can have class members like private, protected, etc.** | **Members of a Java interface are public by default.** |
| **9)Example:**<br>**public abstract class Shape{**<br>**public abstract void draw();**<br>**}** | **Example:**<br>**public interface Drawable{**<br>**void draw();**<br>**}** |
| | |
| | |

**17.Downcasting with instanceof operator**

**18.Package and Access Modifiers**

| Access Modifier | within class | within package | outside package by subclass only | outside package |
|---|---|---|---|---|
| **Private** | **Y** | **N** | **N** | **N** |
| **Default** | **Y** | **Y** | **N** | **N** |
| **Protected** | **Y** | **Y** | **Y** | **N** |
| **Public** | **Y** | **Y** | **Y** | **Y** |

**19.Encapsulation**

Encapsulation in Java is a process of wrapping code and data together into a single unit, for example, a capsule which is mixed of several medicines.

We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.
The Java Bean class is the example of a fully encapsulated class.

**20.Object class**

The Object class is the parent class of all the classes in java by default. In other words, it is the topmost class of java.
The Object class is beneficial if you want to refer any object whose type you don't know. Notice that parent class reference variable can refer the child class object, know as upcasting.

**21.Object Cloning**
**22.Java Array**
**23.Call By Value and Call By Reference**

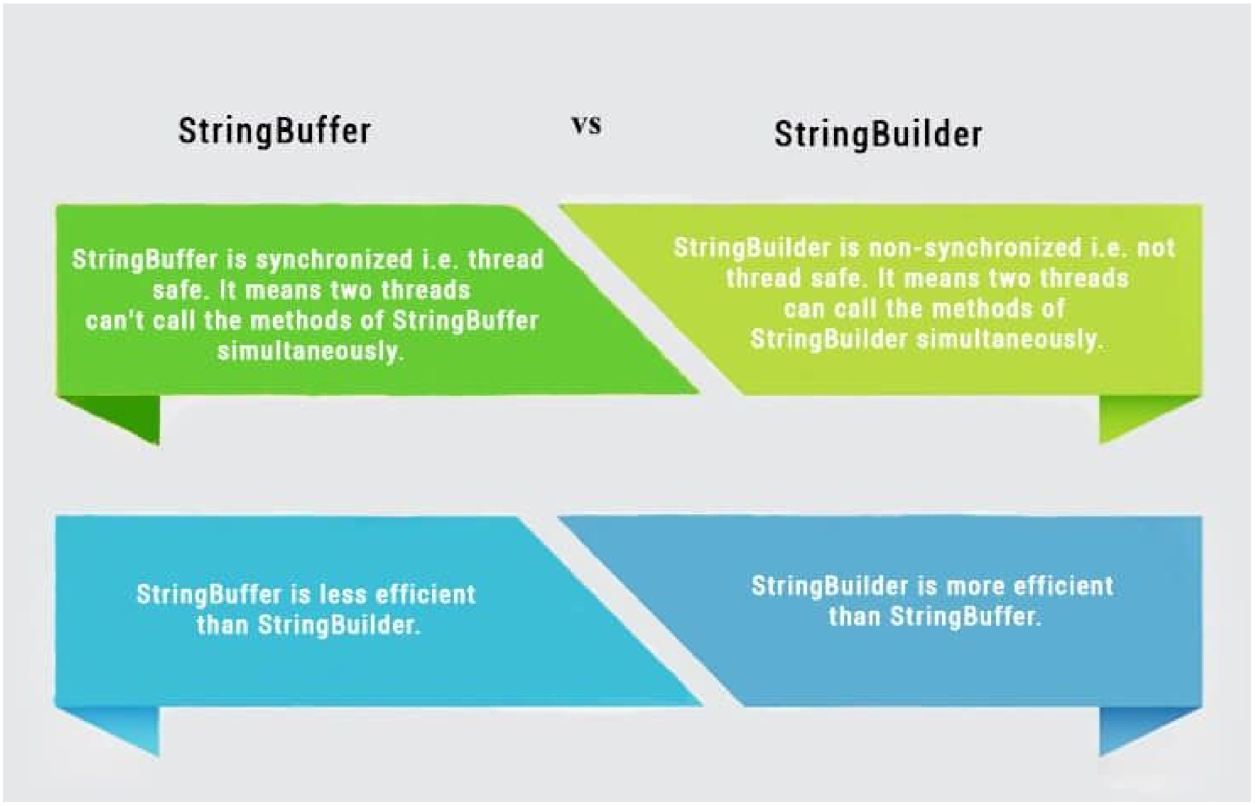
**There is no call by reference in java, call by value is nothing by calling a method by their values not by their references (addresses)**

**24.strictfp keyword**

Java strictfp keyword ensures that you will get the same result on every platform if you perform operations in the floating-point variable. The precision may differ from platform to platform that is why java programming language have provided the strictfp keyword, so that you get same result on every platform. So, now you have better control over the floating-point arithmetic.

**25.Immutable String**

**26.String Comparison**
**27.String Concatenation**
**28.Substring**
**29.Methods of String class**
**30.StringBuffer class vs StringBuilder class**



| StringBuffer | vs | StringBuilder |
|---|---|---|
| StringBuffer is synchronized i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously. | | StringBuilder is non-synchronized i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously. |
| StringBuffer is less efficient than StringBuilder. | | StringBuilder is more efficient than StringBuffer. |

**32.Creating Immutable class**
**33.toString method**
**34.StringTokenizer class**

-

**Sr. No.** **DATA STRUCTUREs**

1 The Collection Interface
This enables you to work with groups of objects; it is at the top of the collections hierarchy.

2 The List Interface
This extends Collection and an instance of List stores an ordered collection of elements.

3 The Set
This extends Collection to handle sets, which must contain unique elements.

4 The SortedSet
This extends Set to handle sorted sets.

5 The Map
This maps unique keys to values.

6 The Map.Entry
This describes an element (a key/value pair) in a map. This is an inner class of Map.

7 The SortedMap
This extends Map so that the keys are maintained in an ascending order.

8 The Enumeration
This is legacy interface defines the methods by which you can enumerate (obtain one at a time) the elements in a collection of objects. This legacy interface has been superceded by Iterator.

HashSet
Extends AbstractSet for use with a hash table.

TreeSet

Implements a set stored in a tree. Extends AbstractSet.

**In this big data era, many companies to try share data We have considered a scenario in which two organizations trying to share their data to a data collector which performs data mining on it and makes money. As the data contains sensitive info they have to provide security. That can be achieved by performing anoninomizations techniques. There are various factors which effect the utilities of organizations. We have solved this problem using Game theory in cyber security and found a nash equilibrium for this scenario.**

**Our work was accepted in IEEE's ICACCE'20 conference .**