**Interview Questions**

1. **What is an Object cloning?**

An ability to recreate an object is completely similar to an existing object is known as object cloning in Java. Java provides clone() method to clone an existing object offering same functionality of the original object.

1. **Difference between the instance and local variables?**

Instance variables are declared inside a class and the scope is limited to only a specific object.

Local variables can be anywhere inside method, or a specific block of code. Also, the scope is limited to the block of code where the variable is declared.

1. **Explain Java String Pool?**

A collection of strings in Java's Heap memory is reffered as a java string pool. In case you try to create a new string object, JVM first checks for the presence of the object in the pool. If available, the same object reference is shared with the variable, else a new object is created.

1. **What is a static method?**

Static method belongs to the class, doesn’t require any object state.  
Accessed directly by specifying its name, static methods called without creating an object of class.  
class Test {  
public static void myStaticMethod () {  
System.out.println(" Hello ");  
}  
public void myPublicMethod() {  
System.out.println(" Hello ");  
}  
public static void main(String[] args) {  
myStaticMethod(); //Hello  
Test test = new Test ();  
test.myPublicMethod();//Hello  
}

**5. What are Spring Beans?**

Spring beans refer to the objects that are managed by the Spring IoC (Inversion of Control) container. These beans are Java objects created, configured, and managed by the spring container. They are defined in the spring configuration files and can be injected into other beans to promote loose coupling.

**6. Describe the differences between Hibernate’s transient, persistent, and detached states?**

New (Transient) State:  
● When you create a new instance of a Java class, it's in the 'New' or 'Transient' state. It's not associated with a Hibernate Session and not saved in the database.  
Persisted (Managed) State:  
● an entity is in the 'Persisted' state when it's associated with a Hibernate Session and represents a row in the database.  
● This happens when you save or retrieve an entity using the Session.  
Detached State:  
● If you close the Hibernate Session, the entity becomes 'Detached'. It's no longer tracked by Hibernate but still represents a row in the database.  
● Changes to a detached entity are not automatically saved to the database.  
OR  
Here is how the states differ:  
Transient. This state describes new objects that are created in Java but not associated with a Hibernate session.  
Persistent. This state describes objects associated with a Hibernate session.  
Detached. This state describes an object that was formerly Persistent and associated with a Hibernate session.

**7. What is Spring ?**

Spring is a dependency injection framework which makes java application loosely coupled. Loosely coupled means components / Elements are less dependant on each other.

**8. What is IOC Container ?**

IoC is a design principle where the flow of control in a program is inverted, i.e., instead of the application controlling the flow of execution, control is inverted to an external framework or container.In traditional programming, a class often controls the flow of a program by calling methods or instantiating objects. In IoC, this control is inverted, and the framework or container takes control, managing the lifecycle of objects and calling their methods.

**9. What is** **Dependency Injection (DI)?**

Dependency Injection is a specific implementation of the Inversion of Control principle.In DI, dependencies (external objects or services that a class needs to perform its functions) are injected into a class rather than the class creating or managing them internally.There are three common types of dependency injection:Constructor Injection: Dependencies are injected through the class constructor.Setter Injection: Dependencies are injected through setter methods of the class.Method Injection: Dependencies are injected through methods of the class.

**10. What is the difference between Array and Collection in Java?**

Arrays are fixed in size that is once we create an array we can not increase or decrease based on our requirements.

The collection is growable in nature and is based on our requirements. We can increase or decrease of size.

With respect to memory, Arrays are not recommended for use.

With respect to memory, collections are recommended for use.

With respect to performance, Arrays are recommended for use.

With respect to performance, collections are not recommended for use.

Arrays can hold only homogeneous data types elements.

Collection can hold both homogeneous and heterogeneous elements.

  **11. What is ArrayList in Java?**

ArrayList is a part of the Java collection framework and it is a class of java.util package. It provides us with dynamic arrays in Java. The main advantages of ArrayList are, if we declare an array then it’s needed to mention the size but in ArrayList, it is not needed to mention the size of ArrayList if you want to mention the size then you can do it.

Array:Int a[]=new int[5];

ArrayList:ArrayList a=new ArrayList();

**12. What is the difference between List and Set in Java?**

The List is an indexed sequence.

The Set is a non-indexed sequence.

The list allows duplicate elements

The set doesn’t allow duplicate elements.

List implementations are ArrayList, LinkedList, Vector, Stack

Set implementations are HashSet, LinkedHashSet.

**13. What is Method Overloading and Method Overriding ?**

  **Method Overloading :**  
 In Java, method overloading occurs when a class has multiple methods with the same name but different parameters (number, type, or order).  
 Example:  
 public class Calculator {    public int add(int a, int b) {        return a + b;    }  
     public double add(double a, double b) {        return a + b;    }}

  **Method Overriding :**  
  In Java, method overriding happens when a subclass provides a specific implementation for a method that is already defined in its superclass.  
 When you have a class (let's call it a 'superclass') with a certain action or behavior, and then you create another class (a 'subclass') that wants to do the same action but in its own way, that's called method overriding in Java.  
 Example:  
 class Animal {    public void makeSound() {        System.out.println("Some generic sound");    }}  
 class Dog extends Animal {    @Override    public void makeSound() {        System.out.println("Bark! Bark!");    }}

**14. Difference between constructor & method?**

**Constructor :** ==> It used for initializing the state of an object when it is created. Constructors are called implicitly when an object is created using the new keyword.  
 ==> The name of the constructor must be the same as the class name. It has no return type, not even void.

 **Method :** ==> It is used to perform actions or operations on objects, and they are explicitly called by name.  
 ==> Can have any name, and it may have a return type, it can be any valid data type or void if the method does not return a value.  
 public class MyClass {    private int value;  
     // Constructor    public MyClass(int value) {        this.value = value;    }  
     // Method    public void printValue() {        System.out.println("Value: " + value);    }}

**15.** **What is Difference between Static, Final and Constant?**

 **Static :** The static keyword is used to define variables and methods that belong to the class itself, rather than to individual objects of the class. This means that there is only one copy of a static variable or method shared among all objects of that class.  
 **Final :** The final keyword indicates that a variable cannot be reassigned once it is initialized. This means that the value of a final variable is fixed throughout the program's execution.  
 **Constant :** A constant is a fixed value that cannot be modified during the program's execution. In Java, constants are typically defined using final variables. However, not all final variables are constants. For instance, a final variable can be modified if it is declared within a method and its value is assigned within a conditional block.  
 Example : In a class of students,\* A static variable could be the total number of students, \* A final variable might be their exam score, and \* A constant could represent the maximum score, which doesn't change.

**16. Tell us something about JIT compiler?**

* JIT stands for Just-In-Time and it is used for improving the performance during run time. It does the task of compiling parts of byte code having similar functionality at the same time thereby reducing the amount of compilation time for the code to run.
* The compiler is nothing but a translator of source code to machine-executable code. But what is special about the JIT compiler? Let us see how it works:
  + First, the Java source code (.java) conversion to byte code (.class) occurs with the help of the javac compiler.
  + Then, the .class files are loaded at run time by JVM and with the help of an interpreter, these are converted to machine understandable code.
  + JIT compiler is a part of JVM. When the JIT compiler is enabled, the JVM analyzes the method calls in the .class files and compiles them to get more efficient and native code. It also ensures that the prioritized method calls are optimized.
  + Once the above step is done, the JVM executes the optimized code directly instead of interpreting the code again. This increases the performance and speed of the execution.

**17. Can you tell the difference between equals() method and equality operator (==) in Java?**

We are already aware of the **(==) equals** operator. That we have used this to compare the equality of the values. But when we talk about the terms of object-oriented programming, we deal with the values in the form of objects. And this object may contain multiple types of data. So using the **(==) operator** does not work in this case. So we need to go with the .**equals() method.**

Both **[(==)** and **.equals()]** primary functionalities are to compare the values, but the secondary functionality is different.

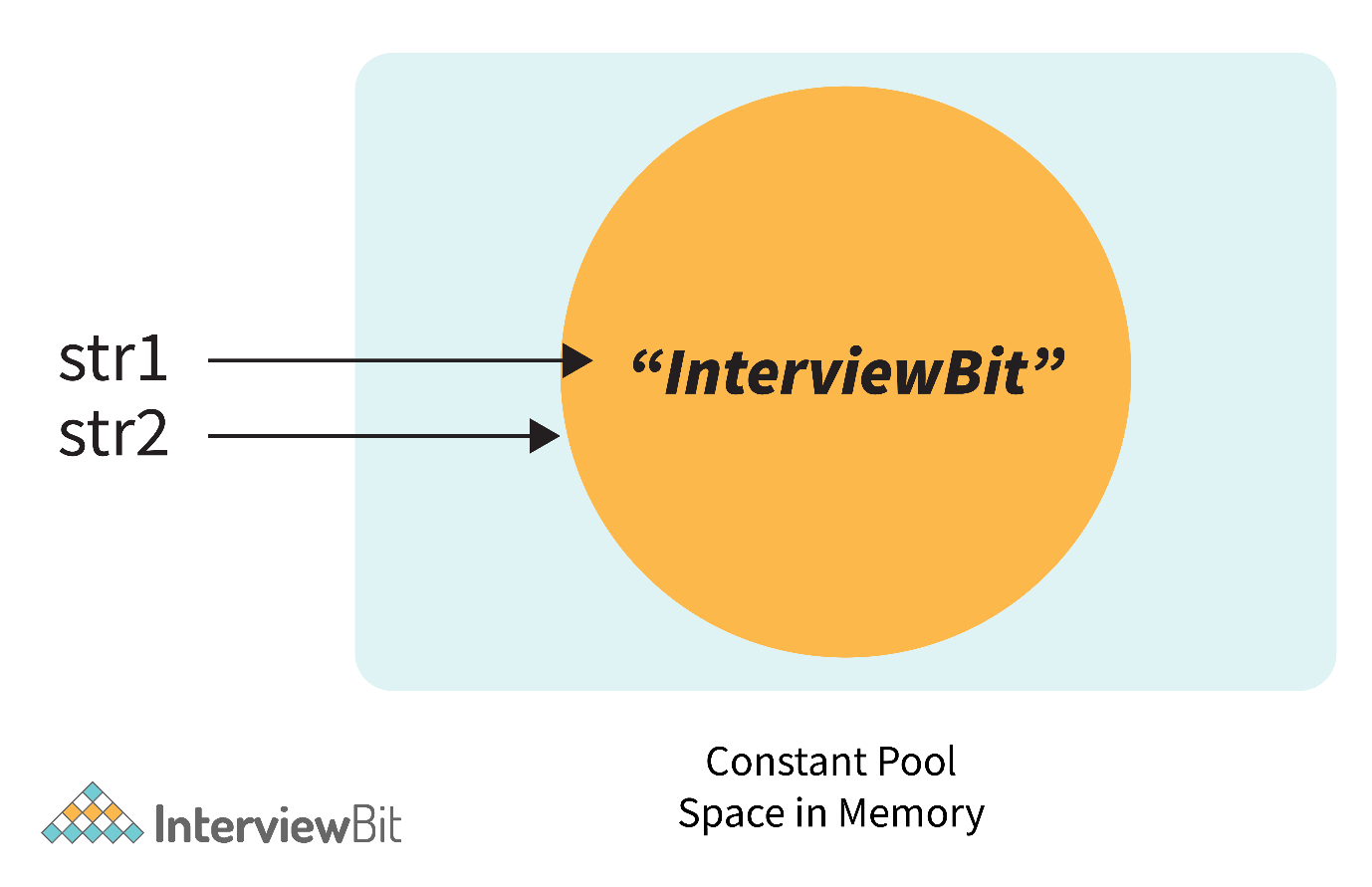
So in order to understand this better, let’s consider this with the example -

String str1 = "InterviewBit";

String str2 = "InterviewBit";

System.out.println(str1 == str2);

This code will print true. We know that both strings are equals so it will print true. But here **(==) Operators** don’t compare each character in this case. It compares the memory location. And because the string uses the constant pool for storing the values in the memory, both **str1** and **str2** are stored at the same memory location. See the detailed Explanation in Question no 73: [**Link**](https://www.interviewbit.com/java-interview-questions/#string-literal-vs-new-string-object-in-java).

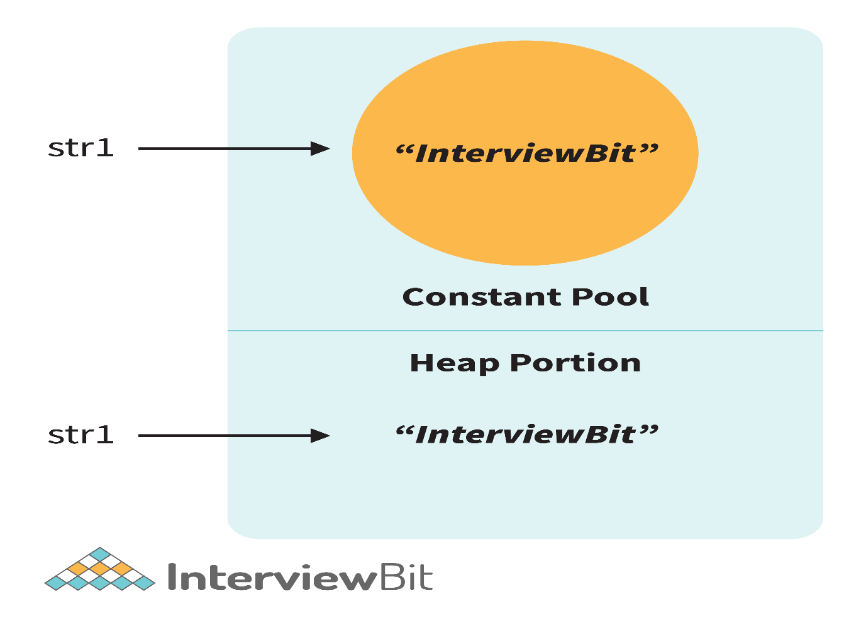


Now, if we modify the program a little bit with -

String str1 = **new** String("InterviewBit");

String str2 = "InterviewBit";

System.out.println(str1 == str2);



Then in this case, it will print false. Because here no longer the constant pool concepts are used. Here, new memory is allocated. So here the memory address is different, therefore ( == ) Operator returns false. But the twist is that the values are the same in both strings. So how to compare the values? Here the .equals() method is used.

**.equals()** method compares the values and returns the result accordingly.  If we modify the above code with -

System.out.println(str1.equals(str2));

Then it returns true.

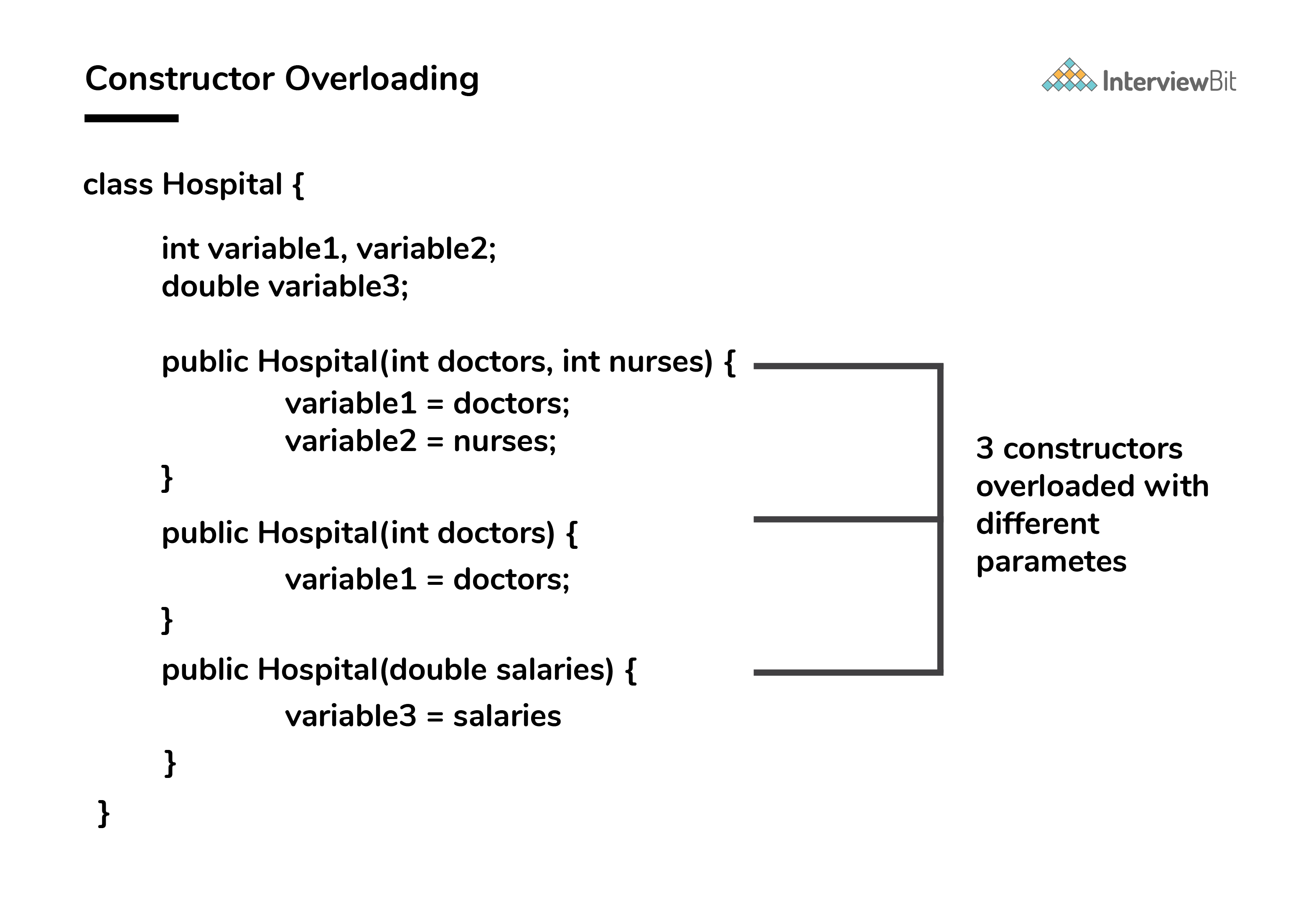
| **equals()** | **==** |
| --- | --- |
| This is a method defined in the Object class. | It is a binary operator in Java. |
| The .equals() Method is present in the Object class, so we can override our custom .equals() method in the custom class, for objects comparison. | It cannot be modified. They always compare the HashCode. |
| This method is used for checking the equality of contents between two objects as per the specified business logic. | This operator is used for comparing addresses (or references), i.e checks if both the objects are pointing to the same memory location. |

**Note:**

* In the cases where the equals method is not overridden in a class, then the class uses the default implementation of the equals method that is closest to the parent class.
* Object class is considered as the parent class of all the java classes. The implementation of the equals method in the Object class uses the == operator to compare two objects. This default implementation can be overridden as per the business logic

**18. Briefly explain the concept of constructor overloading?**

* Constructor overloading is the process of creating multiple constructors in the class consisting of the same name with a difference in the constructor parameters. Depending upon the number of parameters and their corresponding types, distinguishing of the different types of constructors is done by the compiler.
* **class** **Hospital** {
* **int** variable1, variable2;
* **double** variable3;
* **public** **Hospital**(**int** doctors, **int** nurses) {
* variable1 = doctors;
* variable2 = nurses;
* }
* **public** **Hospital**(**int** doctors) {
* variable1 = doctors;
* }
* **public** **Hospital**(**double** salaries) {
* variable3 = salaries
* }
* }



* Three constructors are defined here but they differ on the basis of parameter type and their numbers.

**19. How do u configure a database connection in springboot?**

There are 2 ways to configure a database connection in springboot:

\* Using the application.properties file: This is the most common way to configure a database connection. You can simplify the connection properties in the application.properties file, and springboot will automatically configure the database connection.

\* Using the @ configuration class: this is the more advanced way to configure a database connection. You can create a @Configuration class and specify the connection properties in the class. Springboot will then use the @Configuration class to configure the database connection.

Ex:

@Configuration

public class DatabaseConfiguration {

@Bean

public DataSource dataSource()

DataSourceBuilder dataSourceBuilder = DataSourceBuilder.create();

dataSourceBuilder.url("jdbc:mysql://localhost:3306/mydatabase");

dataSourceBuilder.username("root")

dataSourceBuilder.password("password")

return dataSourceBuilder.build();

}

}

once after configured the database connection, you can use it in your springboot application using Hibernate/JPA or even simpler JDBC template.

**20. What is the purpose of the @SpringBootApplication annotation in a spring boot application?**

The @SpringBootApplication annotation is a convinience annotation that combines 3 annotations spring boot application:

\* @EnableAutoConfiguration: This enables the spring boot autoconfiguration mechanism. Autoconfiguration refers to creating beans automatically by scanning the classpath.

` \* @ComponenScan: This annotation tells the spring boot to scan the current packages and its sub-packages in order to identify annotated classes and configure them as Spring beans.

\* @Configuration: This annotation designates the class as a configuration class for Java configuration.

**21. Can you explain the different types of caching mechanisms available?**

Here are the main types of caching mechanisms:

\*In-Memory Caching : In this mechanism, the cache data is stored in memory, allowing for fast access and retrieval. In springboot, the default cache implimentation is an In Memory cache using a ConcurrentHashMap. This type of caching is suitable for applications where the cached data is not expected to change frequently and can fit within the available memory.

\* Distributed caching: It involves storing cache data accross multiple node in a distributed environment. It allows for scaling the caching capability and provides high availability. Popular distributed casching frameworks used with Spring Boot includes Redis and Hazelcast.

**22. What is constructor chaining?**

Constructor chaining is the process of calling a sequence of constructors. We can do it in two ways: by using this() keyword for chaining constructors in the same class. by using super() keyword for chaining constructors from the parent class.

public class ConstructorChaining {  
  
    ConstructorChaining(){  
        System.out.println("hi");  
    }  
  
    ConstructorChaining(int x) {  
        this();  
        System.out.println(x);  
        System.out.println("Hello");  
    }  
    ConstructorChaining(int z, int y) {  
        this(y);  
        int a = z+y;  
        if (a %2 == 0){  
            System.out.println(a%2);  
        }  
        System.out.println(a);  
        System.out.println("by");  
    }  
  
    public static void main(String[] args) {  
        ConstructorChaining chaining = new ConstructorChaining(1,20);  
  
    }

**23. Can we override static methods?**

We can declare static methods with the same signature in the subclass, but it is not considered overriding as there won’t be any run-time polymorphism. Hence the answer is NO.  
If a child class defines a static method with the same signature as a static method in the parent class, the method in the child class is hidden by the method in the parent class.  
package org.example;

    class Parent {  
        static void display() {  
            System.out.println("Parent's display method");  
        }  
  
        void show(){  
            System.out.println("Parent show method");  
        }  
    }  
  
    class Child extends Parent {  
        static void display() {  
            System.out.println("Child's display method");  
        }  
  
        void show(){  
            System.out.println("Child show method");  
        }  
    }  
  
    public class StaticMethods {  
        public static void main(String[] args) {  
            Parent parent = new Child();  
  
  
            Child.display();  
            parent.show();  
            parent.display();  
  
  
        }  
    }

**24. What is diffrence b/w synchronous & aynchronous api calls?**

synchronous -> We make an api call and wait until server returns the response.  
Other code execution will stop until we get the response.  
  
Aynchronous-> we make an API call and move ahead with rest of the execution rather the waiting for the response.

**25. Can we restrict the visibility of derived method in java?**

No the visibility of derived method must be either same as base method or it can be more then base method lesser visibility then parent method is not allowed.

**26.Explain the use of final keyword in variable, method and class?**

In Java, the final keyword is used as defining something as constant /final and represents the non-access modifier.  
final variable:When a variable is declared as final in Java, the value can’t be modified once it has been assigned.If any value has not been assigned to that variable, then it can be assigned only by the constructor of the class.final method:A method declared as final cannot be overridden by its children's classes.A constructor cannot be marked as final because whenever a class is inherited, the constructors are not inherited. Hence, marking it final doesn't make sense. Java throws compilation error saying - modifier final not allowed herefinal class:No classes can be inherited from the class declared as final. But that final class can extend other classes for its usage.

**27. Do final, finally and finalize keywords have the same function?**

All three keywords have their own utility while programming.  
Final: If any restriction is required for classes, variables, or methods, the final keyword comes in handy. Inheritance of a final class and overriding of a final method is restricted by the use of the final keyword. The variable value becomes fixed after incorporating the final keyword. Example:  
final int a=100; a = 0;  // errorThe second statement will throw an error.  
Finally: It is the block present in a program where all the codes written inside it get executed irrespective of handling of exceptions. Example:  
try {  
 int variable = 5; }  
 catch (Exception exception) { System.out.println("Exception occurred"); }  
 finally { System.out.println("Execution of finally block"); }Finalize: Prior to the garbage collection of an object, the finalize method is called so that the clean-up activity is implemented. Example:  
public static void main(String[] args) { String example = new String("InterviewBit"); example = null; System.gc(); // Garbage collector called  
 }  
 public void finalize() {  
 // Finalize called  
 }

**28. When can you use super keyword?**

The super keyword is used to access hidden fields and overridden methods or attributes of the parent class.Following are the cases when this keyword can be used:Accessing data members of parent class when the member names of the class and its child subclasses are same.To call the default and parameterized constructor of the parent class inside the child class.Accessing the parent class methods when the child classes have overridden them.

**29. What are the features of JUnit?**

Following are the features of JUnit:

* JUnit is an open-source framework.
* Supports automated testing of test suites.
* Provides annotations for identifying the test methods.
* Provides assertions to test expected results or exceptions of the methods under test.
* Provides a platform for running test cases automatically and checking their results and giving feedback.

**30. What is Polymorphism?**

Polymorphism allows objects to be treated as instances of their parent class, even if they are actually instances of a subclass. It includes method overloading and method overriding and the other way we can tell as, An object showing different behaviour at the different stages of its life cycle.

Ex: Compile-time Polymorphism (Method Overloading):  
class MathOperations {      
int add(int a, int b) {          
return a + b;      
}  
     double add(double a, double b) {          
return a + b;    
 }  
}  
  
 Example :  
Runtime Polymorphism (Method Overriding):  
class Animal {      
void makeSound() {          
System.out.println("Some generic sound");    
 }  
}  
 class Cat extends Animal {      
void makeSound() {          
System.out.println("Meow");      
 }  
}

We have 2 type in polymorphism

1. Run time polymorphism.

2. Compile time polymorphism.

**Run time polymorphism:**

* Method declaration getting bounded to its method definition by the JVM at the run time based on object creation.it is also called late binding.
* Once the method declaration getting bound to its method definition, it can be re-bound so it is called Dynamic binding.
* Method overriding is an example for Run time polymorphism.

**Compile time polymorphism:**

* Method declaration getting bound to its method definition by the compiler at the compile based on the arguments passed. It is also called early binding.
* Once the method declaration getting bound to its method definition, it can’t be re-bound so it is called static binding.
* Method overloading is an example for Compile time polymorphism.

**31. Explain The Features Of Interfaces In Java?**

* All The Methods Defined In Interfaces Are Implicitly Abstract Even Though Abstract Modifier Is Not Declared.
* All The Methods In An Interface Are Public Whether They Are Declared As public Or Not.
* Variables Declared Inside An Interface Are By Default public, static And final.
* Interfaces Cannot Be Instantiated.
* We Cannot Declare static Methods Inside the Interface.
* ‘ Implements’ Keyword Is Used To Implement The Interface.
* An Interface Can Extend Any Number Of Interfaces.
* Multiple Inheritances In Java Are Achieved Through Interfaces.

**32.Why is the main method static in java?**

In Java, the main method is declared as static for a specific reason:  
 1. Entry Point:\* The main method serves as the entry point for a Java application.\* The main method is called by the Java Virtual Machine (JVM) to start the execution of a Java program, it needs to be accessible without creating an instance of the class.  
 2. No Object Creation:\* When a Java program starts, there is no object of the class created yet.\* To allow the main method to be called without creating an object of the class, it must be declared as static. A static method belongs to the class rather than a specific instance of the class.  
 3. JVM Invocation:\* The JVM invokes the main method directly on the class, not on any particular object.\* If the main method were not static, the JVM would need to create an instance of the class before calling the main method, which would be unnecessary overhead.

* public class MyApp {    
   // Static main method      
  public static void main(String[] args)  
  {        
   // Code to be executed when the program starts      
  }  
  }

1. **What is the difference between this & super keyword?**

The "this" keyword in Java refers to the current instance of a class and is used to differentiate between instance variables and local variables. The "super" keyword is used to invoke the superclass methods or access the superclass fields from a subclass.

1. **Difference between ArrayList and LinkedList in the java collection framework?**

ArrayList and LinkedList

ArrayList

This class uses a dynamic array to store the elements in it.

This class implements a List interface. Therefore, this acts as a list.

This class works better when the application demands storing the data and accessing it.

LinkedList

This class uses a doubly linked list to store the elements in it.

This class implements both the List interface and the Deque interface. Therefore, it can act as a list and a deque.

This class works better when the application demands manipulation of the stored data

1. **What are the various interfaces used in Java Collections Framework?**

Collection interface

List interface

Set interface

Queue interface

Dequeue interface

Map interface

1. **What is HashMap?**

 HashMap allows us to store key and value pair

where keys should be unique. If you try to insert the duplicate key, it will replace the element of the corresponding key. It is easy to perform operations using the key index like updation, deletion, etc.