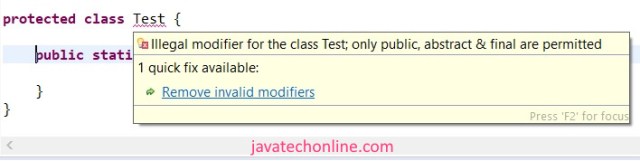
**Why are the Java primitive data types not objects? Why is Java not a pure Object-Oriented programming Language?**

Objects always consume more memory than primitives. If we can solve our purpose without creating more number of objects, the performance will always be better. Java Wrapper Classes were introduced after primitives to make a programmer feel Java as pure Object Oriented Programming language. But still because of the existence of primitives only Java is not a pure Object Oriented Programming Language.

**What is the difference between Access Specifier & Access Modifiers in java? Do we have Access Specifiers in java?**

**Ans:** First of all we should know that there is no term called Access Specifiers in java. Other languages like C, C++ etc. may have defined this term, but java doesn’t. Some of the books & websites refer private, public, protected, default as Access Specifier but Java compiler considers them as Access Modifiers. If we declare any class as private or protected we can check the compiler error by mouse hover on it. The compiler will show “Illegal modifier for the class Test; only public, abstract & final are permitted” as shown below in the snippet. Therefore, all the related keywords like static, abstract, final, synchronized etc. including private, public, protected are Access modifiers in java.

No Access specifiers in Java

**What all access modifiers are allowed in a top level class & in an inner class?**

**Ans :** Most of the times we work with public classes. So there is a high probability that you find this question in the list of Java Interview Questions of the interviewer. Generally they target the area that people ignore, but these concepts come under the category of basic concepts. Let’s see below the list of allowed access modifiers to each one.

**Top level classes ⇒ default, public, abstract, final, strictfp**

**Inner classes       ⇒ default, public, abstract, final, strictfp, private, protected, static**

**What is strictfp modifier? Where is it applied ?**

**Ans:** strictfp modifier was introduced in jdk 1.2. The full form of strictfp is the strict floating point. It is applicable for classes, methods, but not for variables. The arithmetic of floating point numbers generally varies from platform to platform ie. Platform dependent. Suppose we have three platforms where we want to test the value of  an arithmetic 20.0/3 as shown below.

1) for Windows => value may be 6.6666666665  
2) for Linux => value may be 6.6666666  
3) for MAC => value may be 6.666666667

If we want platform independent results for floating point arithmetic, we should use strictfp modifier by declaring it before method. When a method is declared as strictfp then all floating point calculations in that method will follow [IEEE 754 standard](https://en.wikipedia.org/wiki/IEEE_754) and we will get platform independent results. If a class is declared as strictfp, all concrete methods of the class will follow the IEEE 754 standard in the arithmetic calculations.

**♥ strictfp can only be applied with concrete method. Therefore abstract+strictfp combination is illegal for methods.**

**♥ abstract+strictfp is a legal combination for classes as strictfp is applied only to concrete methods.**

**How many types of variables are possible in java? What all access modifiers are allowed with variables?**

**Ans:** Variables are placeholders in java. They are of 4 types in Java:

1. Instance Variables (Non-Static Fields)

2. Class Variables (Static Fields)

3. Local Variables

4. Parameters

♥ Some people don’t consider parameters as a variable, but it comes under the type of variable as per [Oracle java documentation](https://docs.oracle.com/javase/tutorial/java/nutsandbolts/variables.html). Only static, final, transient & volatile modifiers are applicable to variables.

**How can you decide the jdk version of a compiled .class file?**

**Ans:** Use below command to get Major & Minor version of the file:

**javap -verbose ClassNameWithDirectoryPath | find “version”**

On executing this command you will get results something like: minor version: 0 major version: 52 Match your Major.minor version(52.0) from below table. Checking major version will also solve our purpose. In this case this file has been compiled using JDK 1.8

**JDK      Major.minor versions**

**1.4            48.0**

**5 (1.5)      49.0**

**6 (1.6)      50.0**

**7 (1.7)      51.0**

**8 (1.8)      52.0**

**9              53.0**

**10              54.0**

**11              55.0**

**12              56.0**

**13              57.0**

**14              58.0**

**15 59.0**

**What is a local block in java ?**

**Ans :** A nameless block { } that is created inside a method is called a local block. We use it for decreasing the scope of a local variable. We can write any statement inside it what all are applicable to a method. A variable declared inside a local block can’t be accessed after the closing curly bracket of this local block.

**What is the role of a class loader in java ?**

Class Loader’s role is to load classes into the JRE in order to make them available for the JVM. Moreover, Class Loader is a part of the JRE. JVM requests class loader to load the class. Further, class loader tries to find out it in the class path, and if it is not there, it loads the same accordingly. In order to have in-depth knowledge of JVM architecture and Class Loaders, kindly visit our article on [JVM architecture and Class Loaders Java](https://javatechonline.com/jvm-architecture-class-loaders-java/).

**Is the following main method valid in java ?**

Checking main() Method Validity

public class MainMethodValidity {

synchronized final public strictfp static void main(String... str) {

System.out.println("Am I valid ?");

}

}

**Ans:** Yes, this is valid main() method in java. Here, place of an Access modifier doesn’t matter. If we want only one thread can access the main method we can use synchronized keyword in it. If we don’t want another class to extend from this class we can declare it as final. In order to know more about strictfp, check the answer of [this question](https://javatechonline.com/java-interview-questions/#What_is_strictfp_modifier_Where_is_it_applied). Starting from JDK 1.5 we can declare varargs as “String… str” in place of ‘String[ ] args’.

**Why can an abstract method not be declared as private ? Can we use private keyword with abstract keyword ?**

**Ans:** An abstract method can reside either in an abstract class or in an interface. The purpose of declaring an abstract method is to force implementing/child classes to provide a concrete implementation of it. If we want implementing/child classes to provide concrete implementation of it, they should have access to that abstract method. If we declare the method as private, no other classes can access it as private access modifier is limited to the same class. In this case abstract method will never be implemented. So if we use abstract we want to implement it in one side where in other side using private keyword restricts us to access it. Hence, we can’t use abstract & private together.

**Can an Abstract class contain final method ? What is the benefit of having a final method in an abstract class?**

**Ans:** Yes. An abstract class can contain final methods. It is a valid combination, the compiler will not complain. When we don’t need the child class to override a method of it’s parent class, we declare the method as final. In general, we observe this scenario in the Template Method pattern of Java Design patterns. Template method design pattern is to define an algorithm as the skeleton of operations and leave the details to be implemented by the child classes. Parent abstract class contains complete implementation a method and used in the algorithm. This method in base class should be restricted so that the subclasses does not override it. So it is declared as final.

In Template Method pattern, Parent class is known as Template class & a final method as Template method. Let’s make it clear with an example. Suppose we have to implement an algorithm to become an Employee of a company. In this process, there are many small steps to be followed like applying for the job, preparing for the interview, getting an offer letter, joining the company, then someone becomes the employee of the company.

abstract class Employment {

abstract void applyforJob();

abstract void pepareForInterview();

abstract void getOfferLetter();

abstract void joinCompany();

final void becomeEmployee() {

applyforJob();

pepareForInterview();

getOfferLetter();

joinCompany();

}

}

class JavaEmployment extends Employment {

@Override

void applyforJob() {

System.out.println("I had applied for Java Developer in XYZ company");

}

@Override

void pepareForInterview() {

System.out.println("Then I prepared for the interviews");

}

@Override

void getOfferLetter() {

System.out.println("I got the offer letter from XYZ company for Java developer with handsome salary");

}

@Override

void joinCompany() {

System.out.println("I am going to join the XYZ company in next month");

}

}

public class Test {

public static void main(String[] args) {

Employment employment = new JavaEmployment();

employment.becomeEmployee();

}

}

**Output:**

I had applied for Java Developer in XYZ company

Then I prepared for the interviews

I got the offer letter from XYZ company for Java developer with handsome salary

I am going to join the XYZ company in next month

**Can a Final class contain abstract method ?**

**Ans:** No. A final class can’t contain an abstract method because a final class can’t be subclassed and without subclassing we can’t implement the abstract method. Therefore it is totally useless to keep an abstract method inside any final class. Moreover, if we declare a method as abstract in any class, compiler will suggest to make the class abstract to remove compile time errors. If the class is abstract this can’t be declared final as abstract-final combination is illegal in java which is explained in the next question.

**Can we use final & abstract both access modifiers in a method? Can we declare a method as abstract & final both.**

**Ans:** No. It’s illegal contract in java. Abstract will force method to get overridden by child classes whereas final will not allow method to be overridden by child classes. So both are contradictory situation which will not solve the purpose of the method at all. Therefore we can’t declare a method as abstract & final both. These questions on combination of multiple access modifiers are very important. They can be part of Java Interview Questions.

**Is below line a valid class declaration  ?**

**class C implements A extends B {.........}**

**Ans:** No. Its not a valid declaration. The valid declaration is :

**class C extends B implements A {......}**

extends keyword always comes before implements keyword. When we use extends it means we can have methods which are inherited by sub class by default while in case of implements it tells us to override methods forcefully. So we always write first which we already have then others.

**What is the difference between final, finally & finalize() keyword in java ?**

**Ans:** Below is the explanation of all three keywords with complete clarity. Keep your special attention on finally & finalize.

**Final :** Final is a keyword that we use with variable, methods & classes. final variable means it’s value is constant and we can’t reassign a value for that variable. A final method can’t be overridden by a child class. Hence, we can’t create a sub class of a final class.

**Finally :** It is used in context of exception handling with try-catch block to release the resources that we opened in the try block.

**Finalize :** It is a method defined in Object class. Garbage collector calls this method to perform clean up activities before destroying the object from the memory.

**\*♥** Finally meant for cleanup activities related to try block whereas finalize() meant for cleanup activities related to object.

**What is the difference between fully checked & partially checked Exceptions?**

**Ans:** A checked Exception is said to be **fully checked** if and only if all its sub classes are also checked exceptions. eg : IOException, InteruptedException. A checked Exception is said to be **partially checked** if and only if some of its sub classes are unchecked exceptions. eg : Exception, Throwable. The only possible partially checked Exceptions in java are Exception & Throwable. All other exceptions are either fully checked exceptions or unchecked exceptions.

**What are the different useful commands to manage jar files?**

We have different commands to create jar file based on some parameters.

1) **jar  -cvf  test.jar Test.class**  ⇒ To create jar file of only one class

2) **jar  -cvf  test.jar** **Test1.class Test2.class Test3.class**  ⇒  To create jar file of multiple classes

3) **jar  -cvf  test.jar \*.class**  ⇒  To create jar file of all the classes in current path

4) **jar  -cvf  test.jar \*.\*** ⇒  To create jar file of all type of files in current path

5) **jar  -xvf  test.jar** ⇒  To extract(unzip) a jar file

6) **jar  -tvf   test.jar** ⇒  To display name of all the files inside the jar

**What is difference between creating object of a class using new() & newInstance() ?**

**Ans:** When we already know the name of the class whom object to be created we use new operator. When we don’t already know the class name & it is to be decided dynamically at runtime we use newInstance(). Furthermore, we can pass the arguments with the class name using new but in case of newInstance() only no-arg constructor will be called. Syntax for creating objects are as below:

TestClass tc= new TestClass(); TestClass tc= **new** TestClass("test");

Class c=Class.forName("com.dev.SampleClass"); SampleClass s=(SampleClass)c.**newInstance()**;

**What is difference between ClassNotFoundException & NoClassDefFoundError ?**

**Ans: ClassNotFoundException** is a checked Exception. We face this when we try to load class at runtime using Class.forName or the class is not found in the class path. Most of the time, this exception occurs when you try to run an application without updating the class path with required JAR files.

**NoClassDefFoundError** is an unchecked exception. We face this when we create a hard coded class by using new operator. It occurs when required class definition is missing at runtime while it was available at compile time. For example, if we have a method call from a class or accessing any static member of a Class and that class is not available during run-time then JVM will throw NoClassDefFoundError.

**What is covariant return type in java?**

**Ans:** The concept of covariant return type was introduced in jdk 1.5. Before jdk 1.5 in method overriding the overriding method must have the same return type as it’s parent class method has. In this case, overriding method was said to be invariant with respect to return type. But jdk 1.5 onward, the overriding method’s return type can be the sub-type of it’s parent class method’s return type. This is called **covariant return type.** The **covariant return type** always works only for non-primitive return types. In order to understand with an example, [click here](https://javatechonline.com/java-interview-questions/#Is_the_following_code_valid_override).

**What is difference between Method hiding & Method Overriding? What is the purpose of overriding a method?**

**Ans:** Re-defining parent class non-static method in child class is called [method overriding](https://javatechonline.com/java-interview-questions/#Method_Overriding). Re-defining parent class static method in child class is called method hiding. This is because we can’t override parent’s class static methods. If parent’s class method doesn’t solve our purpose, we redefine it in child class. We do it using overriding or hiding technique. For example, below code snippet shows the clear difference.

Method Hiding vs Overriding

class P {

static void m1();

void m2();

void m3();

}

class C extends P{

static void m1(); // method hiding

void m2(); // method overriding

void m4();

}

**What is Method Overloading & Method Overriding in java ?**

**Ans:** This question might appear to be a bit easy question to answer but sometimes some people get confused between them. Even they do not remember the rules on when to use which one.

**Method Overloading :**

Method overloading provides two separate methods in a class with the same name but different arguments, while the method return type may or may not be different, which allows us to reuse the same method name with different arguments. There are some rules to follow while overloading a method. Some of them are mandatory while some are optional as below.

**Mandatory Rules:** Two methods will be treated as overloaded if both follow the mandatory rules below:

1. Both must have the same method name.

2. Both must have different argument lists. If they have same types of arguments but their order of occurrence is different, they will be treated as different arguments.

**Optional Rules:** If both methods follow the above mandatory rules, then they may or may not:

1. Have different access modifiers.

2. Have different return types.

3. Throw different checked or unchecked exceptions.

In general, method overloading happens in the same class. However, a method can also be treated as overloaded in the subclass of that class. This happens because the subclass inherits one version of the method from the parent class and then it can have another overloaded version in it’s definition.

**Method Overriding:**

Whenever we extend a child class, the child class automatically gets all the methods defined in the super class. We call them derived methods. But in some cases, we do not want some derived methods to work in the same manner as they do in the parent class. Then we override them in the child class.

**Mandatory Rules:** The overriding method must follow the following rules:

1) It must have the same method name.

2) It must have the same arguments.

3) It must have the same return type. From Java 5 onward, if overriding method’s return type is subclass of overridden method’s return type, it is valid. This type of relationship is called [covariant return types](https://javatechonline.com/java-interview-questions/#What_is_covariant_return_type_in_java). Both types must be an Object to be a valid override.

4) It must not have a more restrictive access modifier. if parent is public then child as private is not allowed. Access modifiers visibility sequence is : private < default < protected < public which means private is the most restrictive & public is the least restrictive modifier.

5) It must not throw newer or broader checked exceptions.

6) It must be in the child class. Methods can be overridden only in child classes not in the same class.

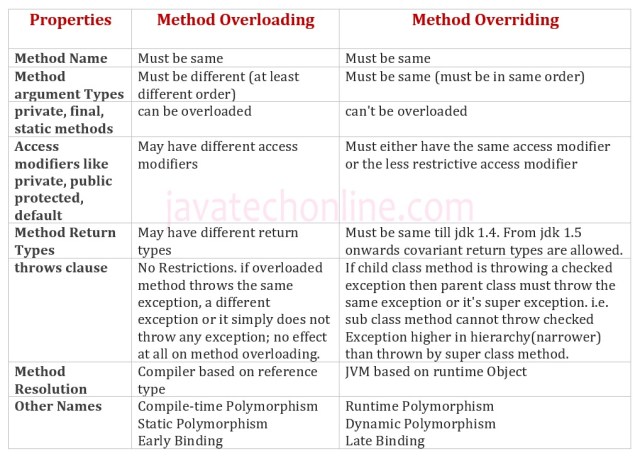
**Optional Rules:** If both methods follow the above mandatory rules, then they :

1. May have a less restrictive access modifier. if parent is protected then child as public is allowed.

2. May throw narrower(below in class hierarchy) checked exceptions or any unchecked exception.

**What is difference between Method Overloading & Method Overriding ?**

**Ans:** Please go through the below table to see clear difference between both. This question has high probability in the list of Java Interview Questions in the interview room.

Method Overloading vs Method Overriding

**Is the following code valid override ?**

class Base {

Number m1() {

return 24;

}

}

class child extends Base{

Short m1() {

return 2;

}

}

**Ans:** This code is valid starting from jdk 1.5 on the basis of new feature [covariant return types](https://javatechonline.com/java-interview-questions/#What_is_covariant_return_type_in_java). Till jdk 1.4 the code was invalid as child class method’s return type doesn’t match with base class method’s return type.

**♥ Note :** Below is the command to check the code in different versions of java. If you have higher version of java, then also you can test the code in lower versions using below command from command prompt. Here I have used jdk 1.4 version as an example.

**javac -source 1.4 Test.java**

**Have you ever faced “java.lang.UnsupportedClassVersionError” while running your java application?**

**Ans:** This runtime error comes when you compiled your java file in higher version of JDK & tried to run this file in lower version of JDK. In contrast, Lower version compiler generated .class files can be run by higher version JVM. But higher version compiler generated .class files can’t be run by lower version JVMs. If we try to run, we will get runtime exception saying “java.lang.UnsupportedClassVersionError”. Major & minor versions represent .class file version. JVM will use these versions to identify which version of compiler generated the current .class file.

**M.m => 'M' stands for major version, 'm' stands for minor version. Below is the versions to identify respective jdk.**

1) java 1.5  ->  49.0  
2) java 1.6  ->  50.0  
3) java 1.7  ->  51.0 and so on…

You will find these version numbers in stack-trace of your error to take further action. For complete list of all versions [visit here](https://javatechonline.com/java-interview-questions/#How_can_you_decide_the_jdk_version_of_a_compiled_class_file).

**What is Magic Value in Java?**

**Ans :** This is a predefined value used by JVM to identify that .class file is generated by valid compiler or not. The first 4 bytes of the class file is magic number. This value should be ‘0xCAFEBAB’. Whenever we execute a java class, if JVM is unable to find valid magic number then we will get runtime exception saying [Exception in thread “main” java.lang.ClassFormatError : Incompatible Magic Value].

**What is ‘var’ identifier introduced in Java 10 ?**

**Ans :** ‘var’ is a case sensitive type identifier that helps java compiler to infer the values, introduced in Java 10 version. However, it is not a keyword. We can only use them in case of local variables. Moreover, we can use var anywhere in java except in class name. For example, below is the list of some valid and invalid var declarations.

Use of 'var' type

var a; //not allowed

var b =5; // allowed

var c ="Hi"; // allowed

var d = true; // allowed

var e = new int[]{}; // allowed

var f = new String(); // allowed

var g = new Test(); // allowed

var h = null; //not allowed

var var = 10; //allowed second var will be infered as just a variable name

int var = 24; // allowed

void var(){} // allowed

class var {} // not Allowed is a restrited type name and cant be defined as a type

class Var () // allowed as var is case sensitive identifier

**How many ways are there to store string data inside a java program ?**

**Ans :** There are four ways to store a string data in java program:

♠ char[]  :  Using Character Array

♦ String  :  Using String class

♦ StringBuffer  : Using StringBuffer

♠ class StringBuilder : Using StringBuilder class

**What is ‘stream’ in Java ? What was the purpose of introducing it ? How does it differ from a collection?**

**Ans:** java.util.stream is a package introduced in jdk 1.8. It has several interfaces & classes to work on streams(flow of data). Classes under stream package introduced to support functional-style operations on the stream of elements. However, please note that Stream is not a collection rather it is a technique to make processing of collection’s data easy by supporting functional-style operations. Furthermore, Streams differs from a collection in several ways.

♠ A stream is not a data structure that stores elements.  
♦ An operation on a stream produces a result, but does not modify its source.  
♥ While collections have a finite size, streams need not.  
♣ The elements of a stream are only visited once during the life of a stream. Like an Iterator, a new stream must be generated to revisit the same elements of the source.

**What is a referenced variable in Java ? How many types of reference variables are possible?**

**Ans:** When we create a variable by using Objects (as a Datatype), we call them referenced variables. However, we create them by using Array, Class, Interface or Enum. Moreover, referenced variables store reference of the object while primitive variables store the values directly. Types of possible referenced variables are : Local Static Non-Static Final Volatile Transient

**How can we restrict a class or interface to be subclassed without using final keyword?**

**Ans :** We can use sealed modifier(a preview feature of JDK 15) in the declaration of class or interface. In the below code snippet Class Child3 is not allowed to extend from class Parent. Only Child1 & Child2 can extend from Parent as they are part of ‘permits’ of the super class Parent. For example, observe the below code. In order to get more detail on the sealed feature of JDK 15, kindly visit the [JDK 15 Features page](https://javatechonline.com/jdk-15-features/).

Sealed modifier of JDK 15 feature

@SuppressWarnings("preview")

sealed class Parent

permits Child1, Child2 { }

class Child3 extends Parent{} // not allowed

@SuppressWarnings("preview")

non-sealed class Child1 extends Parent{ } // allowed

**Is below statement syntactically correct?**

**// A final resource**

**final Resource resource1 = new Resource("resource1");**

**// An effectively final resource**

**Resource resource2 = new Resource("resource2");**

**try (resource1; resource2) {**

**...**

**}**

**Ans :** From this question the interviewer wants to check your knowledge on JDK enhancements  in deferent versions. Now the answer of this question is ” Syntax is correct if we are using JDK 9 + version whereas it is incorrect if we are using JDK 8 and below versions. If you already have a resource as a final or effectively final variable, you can use that variable in a try-with-resources statement without declaring a new variable. An “effectively final” variable is one whose value never changed after it is initialized.

**How to convert Map to List of Objects or POJO ?**

**Ans :** This question is not only important in your day by day programming but also in Java Interview Questions point of view. We will solve this problem using two ways. Let’s first write a POJO class to create objects as below. For example, let’s assume we have an Employee class as below.

**public class Employee {**  
  
 **private Integer** empId**;**  
 **private String** empName**;**  
 **private String** empDept**;**  
  
 **public Employee( Integer** empId**, String** empName**, String** empDept**) {**  
  
 **super();**   
 **this.**empId**=**empId**;**  
 **this.**empName**=** empName**;**  
 **this.**empDept**=**empDept**;**  
  
 **}**   
 **@Override**  
 **public String toString() {**  
 **return "Employee [empId=" +** empId **+ ", empName=" +** empName **+ ", empDept=" +** empDept **+ "]";**  
 **}**  
  
**}**

**Method #1 : Using ArrayList() Constructor**

We will use map’s keySet() method to get all the keys and create an ArrayList to get keyList from them. Likewise, we will use the map’s values() method to get all the values and create an ArrayList to get valueList from them.

**import java.util.\*;**

**public class MapToList{**

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

**Map<Integer, Employee>** map **= new HashMap<>();**

map**.put(101, new Employee(101,"Robert","IT");**

map**.put(102, new Employee(102,"Mary","HR");**

map**.put(103, new Employee(103,"Johnson","Admin");**

map**.put(104, new Employee(104,"William","QA");**

map**.put(105, new Employee(105,"Smith","IT");**

**List<Integer>** keyList **= new ArrayList<Integer>(**map**.keySet());**

**List<Employee>** valueList **= new ArrayList<Employee>(**map**.values());**

**System.out.println("Key List : " +** keyList**);**

**System.out.println("Value List : " +** valueList**);**

**}**

**}**

**Method #2 : Using Stream of JDK 8**

In the below program, instead of using ArrayList constructor, we’ve used stream() to convert the map to a list. Further, We’ve converted the keys and values to stream. Finally, converted it to a list using collect() method by passing Collectors’ toList() as a parameter.

**import java.util.\*;**  
**import java.util.stream.Collectors;**  
  
**public class MapToListUsingStream {**   
 **public static void main(String[]** args**) {**   
 **Map<Integer, Employee>** map **= new HashMap<>();**   
 map**.put(101, new Employee(101,"Robert","IT"));**   
 map**.put(102, new Employee(102,"Mary","HR"));**   
 map**.put(103, new Employee(103,"Johnson","Admin"));**   
 map**.put(104, new Employee(104,"William","QA"));**  
 map**.put(105, new Employee(105,"Smith","IT"));**   
  
 **List<Integer>** keyList **=** map**.keySet().stream().collect(Collectors.toList());**  
 **List<Employee>** valueList **=** map**.values().stream().collect(Collectors.toList());**  
  
 **System.out.println("Key List : " +** keyList**);**   
 **System.out.println("Value List : " +** valueList**);**   
  
 **}**  
**}**

The output of both the programs will be the same.

**What are the three usages of static keyword?**

This question may be a part of ‘Java Interview Questions’ to check in-depth knowledge of the static keyword. However, most of the people immediately tell the purpose of static keyword something like ‘to restrict a class to share the same values of member variables among all instances’. Everyone knows the three usage but sometimes it slips from the mind. Below are the three usages.

1) When you want to store data that’s common to all instances.  
2) When you need to access class members without creating an object of the class.  
3) When you want to execute a block of code before calling any method of the class ie. Static block.

**What do you think ‘Is Optional the Same as null?’**

Optional is a new class under package java.util. The purpose of this call is to address the infamous NullPointerException. Before Java 8, programmers would return null instead of [Optional](https://javatechonline.com/java-8-features/#Optional). There were a few drawback with this approach. One was that there wasn’t a clear way to express that null might be a special value. By contrast, returning an Optional is a clear statement in the API that there might not be a value in there.  
Another advantage of Optional is that you can use a functional programming style with ifPresent() and the other methods rather than needing an if statement.

## ava Basic Interview Questions

### 1. Why is Java a platform independent language?

Java language was developed in such a way that it does not depend on any hardware or software due to the fact that the compiler compiles the code and then converts it to platform-independent byte code which can be run on multiple systems.

* The only condition to run that byte code is for the machine to have a runtime environment (JRE) installed in it.

### 2. Why is Java not a pure object oriented language?

Java supports primitive data types - byte, boolean, char, short, int, float, long, and double and hence it is not a pure object-oriented language.

### 3. Pointers are used in C/ C++. Why does Java not make use of pointers?

Pointers are quite complicated and unsafe to use by beginner programmers. Java focuses on code simplicity, and the usage of pointers can make it challenging. Pointer utilization can also cause potential errors. Moreover, security is also compromised if pointers are used because the users can directly access memory with the help of pointers.

Thus, a certain level of abstraction is furnished by not including pointers in Java. Moreover, the usage of pointers can make the procedure of garbage collection quite slow and erroneous. Java makes use of references as these cannot be manipulated, unlike pointers.

You can download a PDF version of Java Interview Questions.

[Download PDF](javascript:void(0))

### 4. What do you understand by an instance variable and a local variable?

**Instance variables** are those variables that are accessible by all the methods in the class. They are declared outside the methods and inside the class. These variables describe the properties of an object and remain bound to it at any cost.

All the objects of the class will have their copy of the variables for utilization. If any modification is done on these variables, then only that instance will be impacted by it, and all other class instances continue to remain unaffected.

**Example:**

class Athlete {

public String athleteName;

public double athleteSpeed;

public int athleteAge;

}

**Local variables** are those variables present within a block, function, or constructor and can be accessed only inside them. The utilization of the variable is restricted to the block scope. Whenever a local variable is declared inside a method, the other class methods don’t have any knowledge about the local variable.

**Example:**

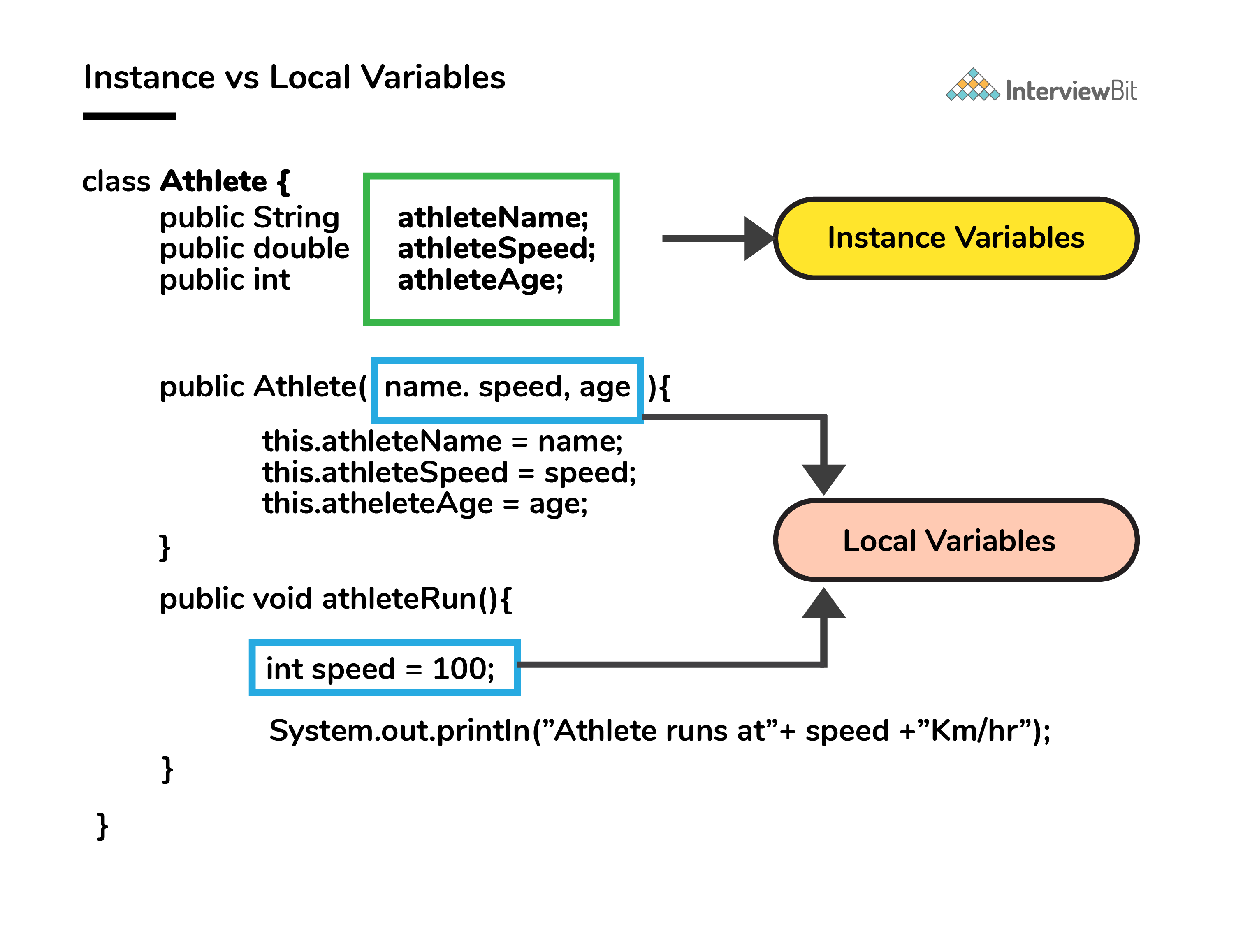
public void athlete() {

String athleteName;

double athleteSpeed;

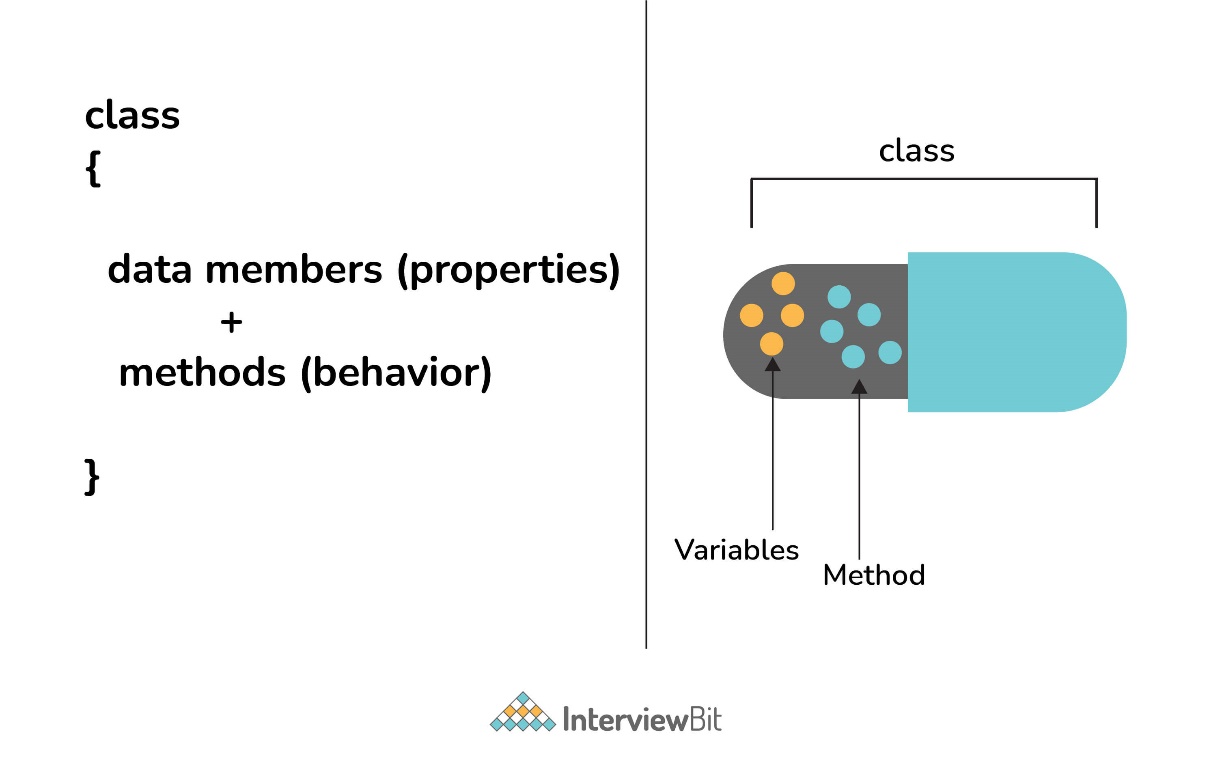
int athleteAge;

}



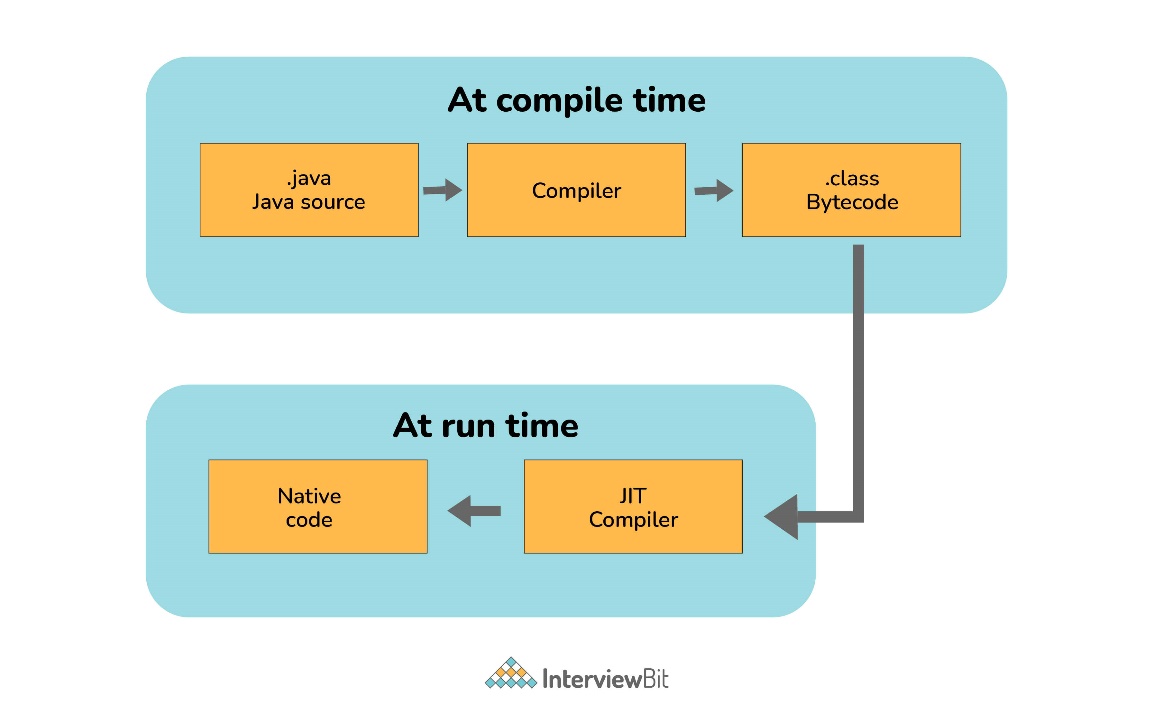
### 5. What do you mean by data encapsulation?

* Data Encapsulation is an [Object-Oriented Programming](https://www.interviewbit.com/oops-interview-questions/) concept of hiding the data attributes and their behaviors in a single unit.
* It helps developers to follow modularity while developing software by ensuring that each object is independent of other objects by having its own methods, attributes, and functionalities.
* It is used for the security of the private properties of an object and hence serves the purpose of data hiding.



### 6. 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.



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

| **equals()** | **==** |
| --- | --- |
| This is a method defined in the Object class. | It is a binary operator in Java. |
| 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.

### 8. How is an infinite loop declared in Java?

Infinite loops are those loops that run infinitely without any breaking conditions. Some examples of consciously declaring infinite loop is:

* Using For Loop:

for (;;)

{

// Business logic

// Any break logic

}

* Using while loop:

while(true){

// Business logic

// Any break logic

}

* Using do-while loop:

do{

// Business logic

// Any break logic

}while(true);

### 9. 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;

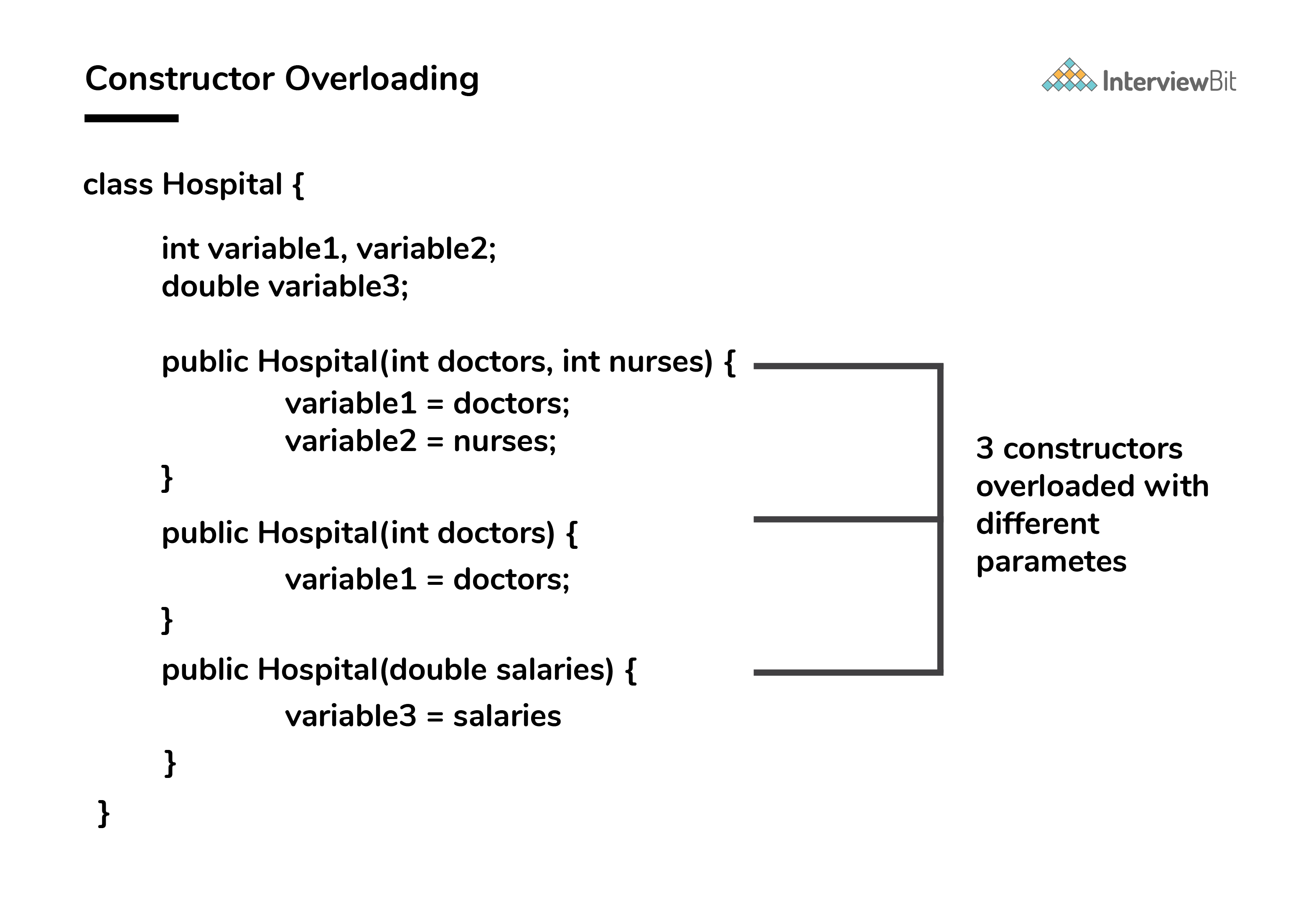
}

public Hospital(double salaries) {

variable3 = salaries

}

}



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

### 10. Comment on method overloading and overriding by citing relevant examples.

In Java, **method overloading** is made possible by introducing different methods in the same class consisting of the same name. Still, all the functions differ in the number or type of parameters. It takes place inside a class and enhances program readability.

The only difference in the return type of the method does not promote method overloading. The following example will furnish you with a clear picture of it.

class OverloadingHelp {

public int findarea (int l, int b) {

int var1;

var1 = l \* b;

return var1;

}

public int findarea (int l, int b, int h) {

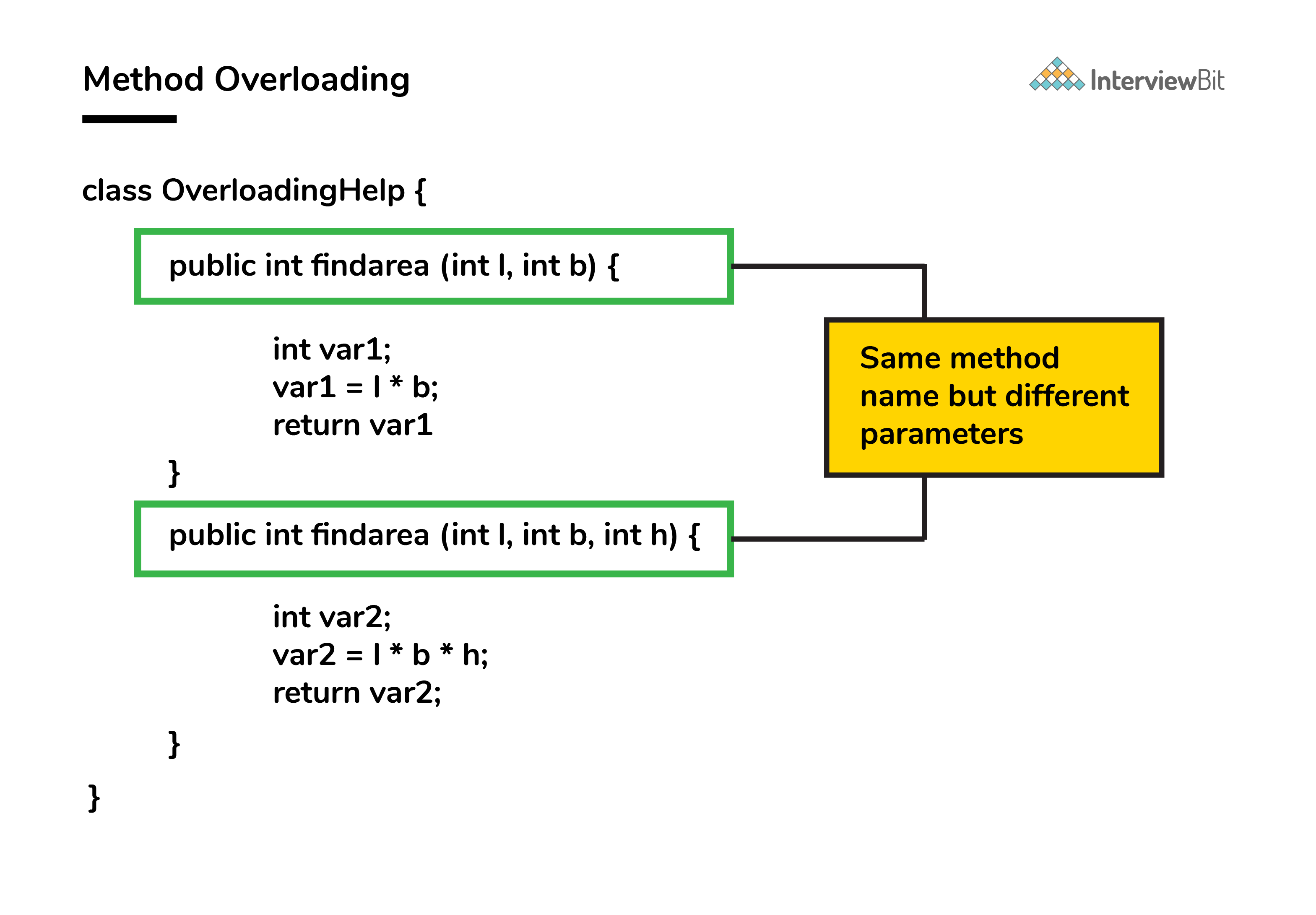
int var2;

var2 = l \* b \* h;

return var2;

}

}



Both the functions have the same name but differ in the number of arguments. The first method calculates the area of the rectangle, whereas the second method calculates the area of a cuboid.

**Method overriding** is the concept in which two methods having the same method signature are present in two different classes in which an inheritance relationship is present. A particular method implementation (already present in the base class) is possible for the derived class by using method overriding.  
Let’s give a look at this example:

class HumanBeing {

public int walk (int distance, int time) {

int speed = distance / time;

return speed;

}

}

class Athlete extends HumanBeing {

public int walk(int distance, int time) {

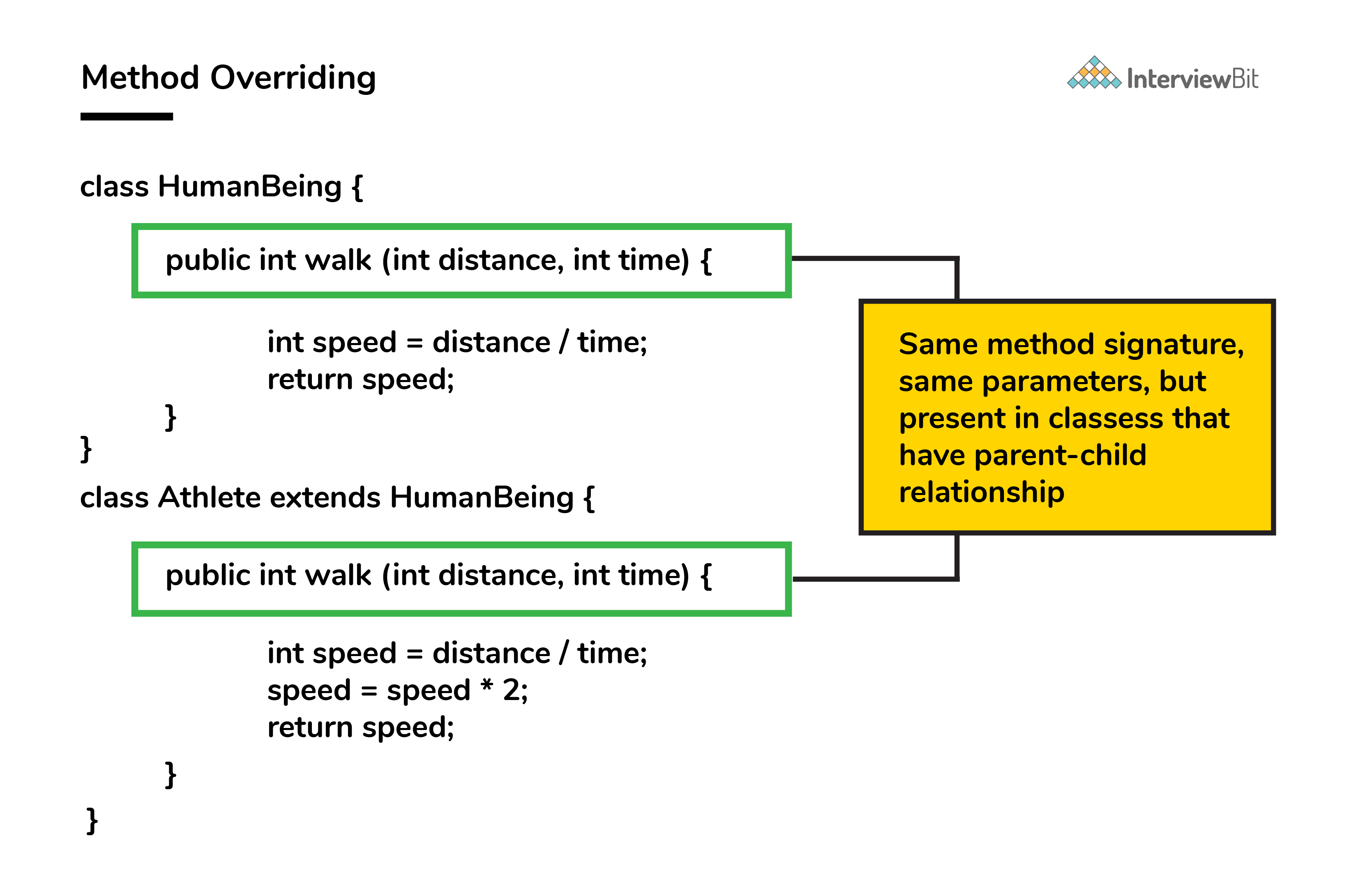
int speed = distance / time;

speed = speed \* 2;

return speed;

}

}



Both class methods have the name walk and the same parameters, distance, and time. If the derived class method is called, then the base class method walk gets overridden by that of the derived class.

### 11. A single try block and multiple catch blocks can co-exist in a Java Program. Explain.

Yes, multiple catch blocks can exist but specific approaches should come prior to the general approach because only the first catch block satisfying the catch condition is executed. The given code illustrates the same:

public class MultipleCatch {

public static void main(String args[]) {

try {

int n = 1000, x = 0;

int arr[] = new int[n];

for (int i = 0; i <= n; i++) {

arr[i] = i / x;

}

}

catch (ArrayIndexOutOfBoundsException exception) {

System.out.println("1st block = ArrayIndexOutOfBoundsException");

}

catch (ArithmeticException exception) {

System.out.println("2nd block = ArithmeticException");

}

catch (Exception exception) {

System.out.println("3rd block = Exception");

}

}

}

Here, the second catch block will be executed because of division by 0 (i / x). In case x was greater than 0 then the first catch block will execute because for loop runs till i = n and array index are till n-1.

### 12. 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 here
* **final class:**
  + No classes can be inherited from the class declared as final. But that final class can extend other classes for its usage.

### 13. 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; // error

The 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

}

### 14. 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.
* The following example demonstrates all 3 cases when a super keyword is used.

public class Parent{

private int num = 1;

Parent(){

System.out.println("Parent class default constructor.");

}

Parent(String x){

System.out.println("Parent class parameterised constructor.");

}

public void foo(){

System.out.println("Parent class foo!");

}

}

public class Child extends Parent{

private int num = 2;

Child(){

System.out.println("Child class default Constructor");

super(); // to call default parent constructor

super("Call Parent"); // to call parameterised constructor.

}

void printNum(){

System.out.println(num);

System.out.println(super.num); //prints the value of num of parent class

}

@Override

public void foo(){

System.out.println("Parent class foo!");

super.foo(); //Calls foo method of Parent class inside the Overriden foo method of Child class.

}

}

### 15. Can the static methods be overloaded?

Yes! There can be two or more static methods in a class with the same name but differing input parameters.

### 16. Can the static methods be overridden?

* No! Declaration of static methods having the same signature can be done in the subclass but run time polymorphism can not take place in such cases.
* Overriding or dynamic polymorphism occurs during the runtime, but the static methods are loaded and looked up at the compile time statically. Hence, these methods cant be overridden.

### 17. What is the main objective of garbage collection?

The main objective of this process is to free up the memory space occupied by the unnecessary and unreachable objects during the Java program execution by deleting those unreachable objects.

* This ensures that the memory resource is used efficiently, but it provides no guarantee that there would be sufficient memory for the program execution.

### 18. What part of memory - Stack or Heap - is cleaned in garbage collection process?

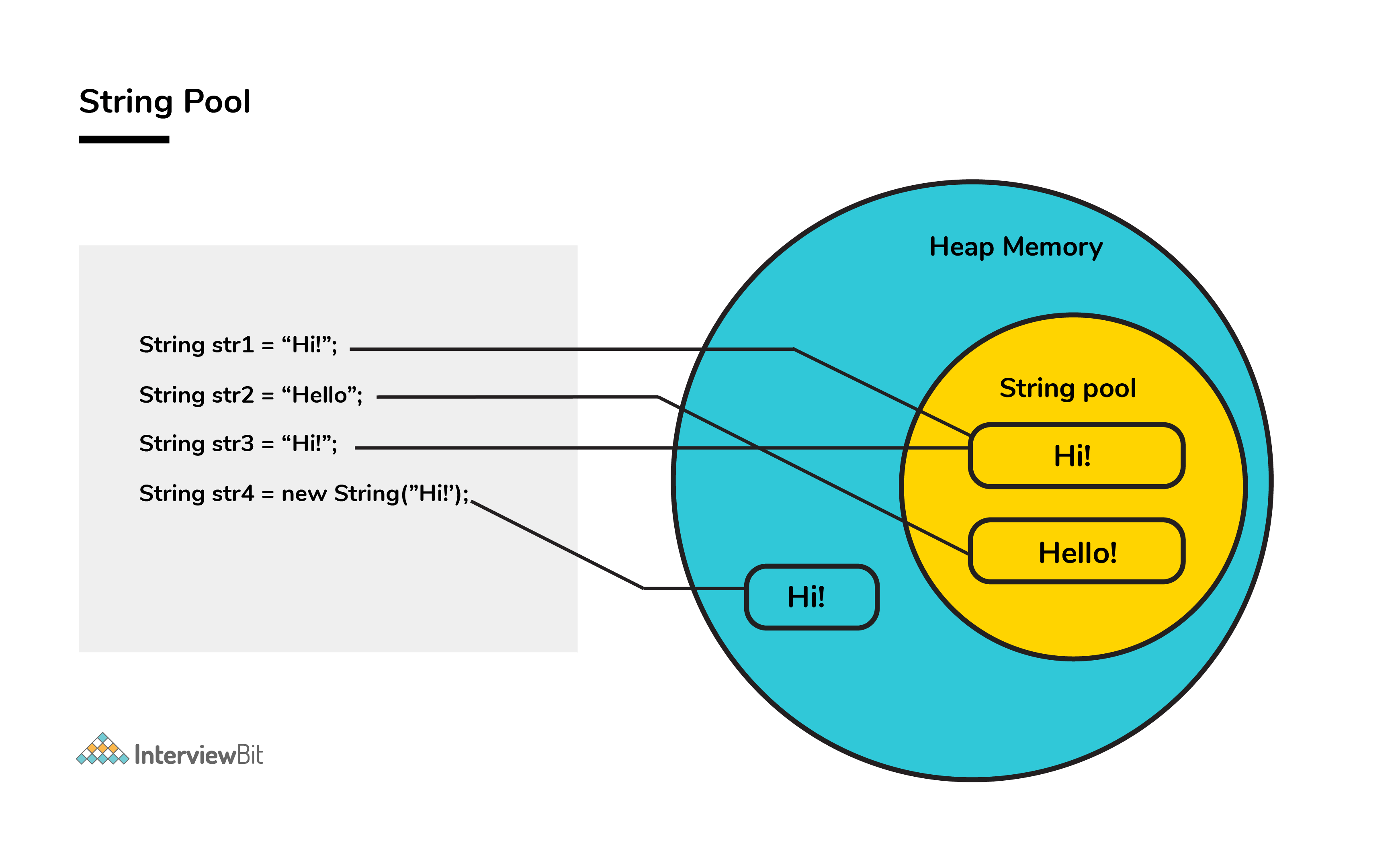
Heap.

## Java Intermediate Interview Questions

### 19. Apart from the security aspect, what are the reasons behind making strings immutable in Java?

A String is made immutable due to the following reasons:

* **String Pool:** Designers of Java were aware of the fact that String data type is going to be majorly used by the programmers and developers. Thus, they wanted optimization from the beginning. They came up with the notion of using the String pool (a storage area in Java heap) to store the String literals. They intended to decrease the temporary String object with the help of sharing. An immutable class is needed to facilitate sharing. The sharing of the mutable structures between two unknown parties is not possible. Thus, immutable Java String helps in executing the concept of String Pool.



* **Multithreading:** The safety of threads regarding the String objects is an important aspect in Java. No external synchronization is required if the String objects are immutable. Thus, a cleaner code can be written for sharing the String objects across different threads. The complex process of concurrency is facilitated by this method.
* **Collections:** In the case of Hashtables and HashMaps, keys are String objects. If the String objects are not immutable, then it can get modified during the period when it resides in the HashMaps. Consequently, the retrieval of the desired data is not possible. Such changing states pose a lot of risks. Therefore, it is quite safe to make the string immutable.

### 20. How would you differentiate between a String, StringBuffer, and a StringBuilder?

* **Storage area:** In string, the String pool serves as the storage area. For StringBuilder and StringBuffer, heap memory is the storage area.
* **Mutability:** A String is immutable, whereas both the StringBuilder and StringBuffer are mutable.
* **Efficiency:** It is quite slow to work with a String. However, StringBuilder is the fastest in performing operations. The speed of a StringBuffer is more than a String and less than a StringBuilder. (For example appending a character is fastest in StringBuilder and very slow in String because a new memory is required for the new String with appended character.)
* **Thread-safe:** In the case of a threaded environment, StringBuilder and StringBuffer are used whereas a String is not used. However, StringBuilder is suitable for an environment with a single thread, and a StringBuffer is suitable for multiple threads.  
  **Syntax:**

// String

String first = "InterviewBit";

String second = new String("InterviewBit");

// StringBuffer

StringBuffer third = new StringBuffer("InterviewBit");

// StringBuilder

StringBuilder fourth = new StringBuilder("InterviewBit");

### 21. Using relevant properties highlight the differences between interfaces and abstract classes.

* **Availability of methods:** Only abstract methods are available in interfaces, whereas non-abstract methods can be present along with abstract methods in abstract classes.
* **Variable types**: Static and final variables can only be declared in the case of interfaces, whereas abstract classes can also have non-static and non-final variables.
* **Inheritance:** Multiple inheritances are facilitated by interfaces, whereas abstract classes do not promote multiple inheritances.
* **Data member accessibility:** By default, the class data members of interfaces are of the public- type. Conversely, the class members for an abstract class can be protected or private also.
* **Implementation:** With the help of an abstract class, the implementation of an interface is easily possible. However, the converse is not true;

**Abstract class example:**

public abstract class Athlete {

public abstract void walk();

}

**Interface example:**

public interface Walkable {

void walk();

}

### 22. In Java, static as well as private method overriding is possible. Comment on the statement.

The statement in the context is completely False. The static methods have no relevance with the objects, and these methods are of the class level. In the case of a child class, a static method with a method signature exactly like that of the parent class can exist without even throwing any compilation error.

The phenomenon mentioned here is popularly known as method hiding, and overriding is certainly not possible. Private method overriding is unimaginable because the visibility of the private method is restricted to the parent class only. As a result, only hiding can be facilitated and not overriding.

### 23. What makes a HashSet different from a TreeSet?

Although both HashSet and TreeSet are not synchronized and ensure that duplicates are not present, there are certain properties that distinguish a HashSet from a TreeSet.

* **Implementation:** For a HashSet, the hash table is utilized for storing the elements in an unordered manner. However, TreeSet makes use of the red-black tree to store the elements in a sorted manner.
* **Complexity/ Performance:** For adding, retrieving, and deleting elements, the time amortized complexity is O(1) for a HashSet. The time complexity for performing the same operations is a bit higher for TreeSet and is equal to O(log n). Overall, the performance of HashSet is faster in comparison to TreeSet.
* **Methods:** hashCode() and equals() are the methods utilized by HashSet for making comparisons between the objects. Conversely, compareTo() and compare() methods are utilized by TreeSet to facilitate object comparisons.
* **Objects type:** Heterogeneous and null objects can be stored with the help of HashSet. In the case of a TreeSet, runtime exception occurs while inserting heterogeneous objects or null objects.

### 24. Why is the character array preferred over string for storing confidential information?

In Java, a string is basically immutable i.e. it cannot be modified. After its declaration, it continues to stay in the string pool as long as it is not removed in the form of garbage. In other words, a string resides in the heap section of the memory for an unregulated and unspecified time interval after string value processing is executed.

As a result, vital information can be stolen for pursuing harmful activities by hackers if a memory dump is illegally accessed by them. Such risks can be eliminated by using mutable objects or structures like character arrays for storing any variable. After the work of the character array variable is done, the variable can be configured to blank at the same instant. Consequently, it helps in saving heap memory and also gives no chance to the hackers to extract vital data.

### 25. What are the differences between JVM, JRE and JDK in Java?

| **Criteria** | **JDK** | **JRE** | **JVM** |
| --- | --- | --- | --- |
| **Abbreviation** | Java Development Kit | Java Runtime Environment | Java Virtual Machine |
| **Definition** | JDK is a complete software development kit for developing Java applications. It comprises JRE, JavaDoc, compiler, debuggers, etc. | JRE is a software package providing Java class libraries, JVM and all the required components to run the Java applications. | JVM is a platform-dependent, abstract machine comprising of 3 specifications - document describing the JVM implementation requirements, computer program meeting the JVM requirements and instance object for executing the Java byte code and provide the runtime environment for execution. |
| **Main Purpose** | JDK is mainly used for code development and execution. | JRE is mainly used for environment creation to execute the code. | JVM provides specifications for all the implementations to JRE. |
| **Tools provided** | JDK provides tools like compiler, debuggers, etc for code development | JRE provides libraries and classes required by JVM to run the program. | JVM does not include any tools, but instead, it provides the specification for implementation. |
| **Summary** | JDK = (JRE) + Development tools | JRE = (JVM) + Libraries to execute the application | JVM = Runtime environment to execute Java byte code. |

### 26. What are the differences between HashMap and HashTable in Java?

| **HashMap** | **HashTable** |
| --- | --- |
| HashMap is not synchronized thereby making it better for non-threaded applications. | HashTable is synchronized and hence it is suitable for threaded applications. |
| Allows only one null key but any number of null in the values. | This does not allow null in both keys or values. |
| Supports order of insertion by making use of its subclass LinkedHashMap. | Order of insertion is not guaranteed in HashTable. |

### 27. What is the importance of reflection in Java?

* The term reflection is used for describing the inspection capability of a code on other code either of itself or of its system and modify it during runtime.
* Consider an example where we have an object of unknown type and we have a method ‘fooBar()’ which we need to call on the object. The static typing system of Java doesn't allow this method invocation unless the type of the object is known beforehand. This can be achieved using reflection which allows the code to scan the object and identify if it has any method called “fooBar()” and only then call the method if needed.

Method methodOfFoo = fooObject.getClass().getMethod("fooBar", null);

methodOfFoo.invoke(fooObject, null);

* Using reflection has its own cons:
  + Speed — Method invocations due to reflection are about three times slower than the direct method calls.
  + Type safety — When a method is invoked via its reference wrongly using reflection, invocation fails at runtime as it is not detected at compile/load time.
  + Traceability — Whenever a reflective method fails, it is very difficult to find the root cause of this failure due to a huge stack trace. One has to deep dive into the invoke() and proxy() method logs to identify the root cause.
* Hence, it is advisable to follow solutions that don't involve reflection and use this method as a last resort.

### 28. What are the different ways of threads usage?

* We can define and implement a thread in java using two ways:
  + **Extending the Thread class**

class InterviewBitThreadExample extends Thread{

public void run(){

System.out.println("Thread runs...");

}

public static void main(String args[]){

InterviewBitThreadExample ib = new InterviewBitThreadExample();

ib.start();

}

}

* **Implementing the Runnable interface**

class InterviewBitThreadExample implements Runnable{

public void run(){

System.out.println("Thread runs...");

}

public static void main(String args[]){

Thread ib = new Thread(new InterviewBitThreadExample());

ib.start();

}

}

* Implementing a thread using the method of Runnable interface is more preferred and advantageous as Java does not have support for multiple inheritances of classes.
* start() method is used for creating a separate call stack for the thread execution. Once the call stack is created, JVM calls the run() method for executing the thread in that call stack.

### 29. What are the differences between constructor and method of a class in Java?

| **Constructor** | **Method** |
| --- | --- |
| Constructor is used for initializing the object state. | Method is used for exposing the object's behavior. |
| Constructor has no return type. | Method should have a return type. Even if it does not return anything, return type is void. |
| Constructor gets invoked implicitly. | Method has to be invoked on the object explicitly. |
| If the constructor is not defined, then a default constructor is provided by the java compiler. | If a method is not defined, then the compiler does not provide it. |
| The constructor name should be equal to the class name. | The name of the method can have any name or have a class name too. |
| 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 here | A method can be defined as final but it cannot be overridden in its subclasses. |
| Final variable instantiations are possible inside a constructor and the scope of this applies to the whole class and its objects. | A final variable if initialised inside a method ensures that the variable cant be changed only within the scope of that method. |

### 30. Java works as “pass by value” or “pass by reference” phenomenon?

Java always works as a “pass by value”. There is nothing called a “pass by reference” in Java. However, when the object is passed in any method, the address of the value is passed due to the nature of object handling in Java. When an object is passed, a copy of the reference is created by Java and that is passed to the method. The objects point to the same memory location. 2 cases might happen inside the method:

* **Case 1:** When the object is pointed to another location: In this case, the changes made to that object do not get reflected the original object before it was passed to the method as the reference points to another location.

For example:

class InterviewBitTest{

int num;

InterviewBitTest(int x){

num = x;

}

InterviewBitTest(){

num = 0;

}

}

class Driver {

public static void main(String[] args)

{

//create a reference

InterviewBitTest ibTestObj = new InterviewBitTest(20);

//Pass the reference to updateObject Method

updateObject(ibTestObj);

//After the updateObject is executed, check for the value of num in the object.

System.out.println(ibTestObj.num);

}

public static void updateObject(InterviewBitTest ibObj)

{

// Point the object to new reference

ibObj = new InterviewBitTest();

// Update the value

ibObj.num = 50;

}

}

Output:

20

* **Case 2:** When object references are not modified: In this case, since we have the copy of reference the main object pointing to the same memory location, any changes in the content of the object get reflected in the original object.

For example:

class InterviewBitTest{

int num;

InterviewBitTest(int x){

num = x;

}

InterviewBitTest(){

num = 0;

}

}

class Driver{

public static void main(String[] args)

{

//create a reference

InterviewBitTest ibTestObj = new InterviewBitTest(20);

//Pass the reference to updateObject Method

updateObject(ibTestObj);

//After the updateObject is executed, check for the value of num in the object.

System.out.println(ibTestObj.num);

}

public static void updateObject(InterviewBitTest ibObj)

{

// no changes are made to point the ibObj to new location

// Update the value of num

ibObj.num = 50;

}

}

Output:

50

### 31. Which among String or String Buffer should be preferred when there are lot of updates required to be done in the data?

StringBuffer is mutable and dynamic in nature whereas String is immutable. Every updation / modification of String creates a new String thereby overloading the string pool with unnecessary objects. Hence, in the cases of a lot of updates, it is always preferred to use StringBuffer as it will reduce the overhead of the creation of multiple String objects in the string pool.

### 32. How to not allow serialization of attributes of a class in Java?

* In order to achieve this, the attribute can be declared along with the usage of transient keyword as shown below:

public class InterviewBitExample {

private transient String someInfo;

private String name;

private int id;

// :

// Getters setters

// :

}

* In the above example, all the fields except someInfo can be serialized.

### 33. What happens if the static modifier is not included in the main method signature in Java?

There wouldn't be any compilation error. But then the program is run, since the JVM cant map the main method signature, the code throws “NoSuchMethodError” error at the runtime.

### 34. What happens if there are multiple main methods inside one class in Java?

The program can't compile as the compiler says that the method has been already defined inside the class.

### 35. What do you understand by Object Cloning and how do you achieve it in Java?

* It is the process of creating an exact copy of any object. In order to support this, a java class has to implement the Cloneable interface of java.lang package and override the clone() method provided by the Object class the syntax of which is:

protected Object clone() throws CloneNotSupportedException{

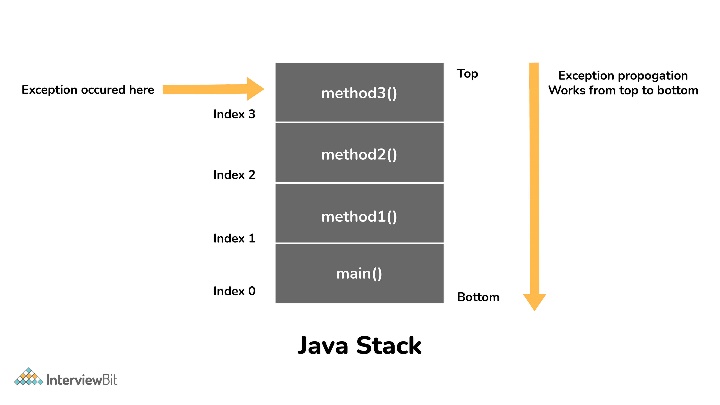
return (Object)super.clone();

}

* In case the Cloneable interface is not implemented and just the method is overridden, it results in CloneNotSupportedException in Java.

### 36. How does an exception propagate in the code?

When an exception occurs, first it searches to locate the matching catch block. In case, the matching catch block is located, then that block would be executed. Else, the exception propagates through the method call stack and goes into the caller method where the process of matching the catch block is performed. This propagation happens until the matching catch block is found. If the match is not found, then the program gets terminated in the main method.



### 37. Is it mandatory for a catch block to be followed after a try block?

No, it is not necessary for a catch block to be present after a try block. - A try block should be followed either by a catch block or by a finally block. If the exceptions likelihood is more, then they should be declared using the throws clause of the method.

### 38. Will the finally block get executed when the return statement is written at the end of try block and catch block as shown below?

public int someMethod(int i){

try{

//some statement

return 1;

}catch(Exception e){

//some statement

return 999;

}finally{

//finally block statements

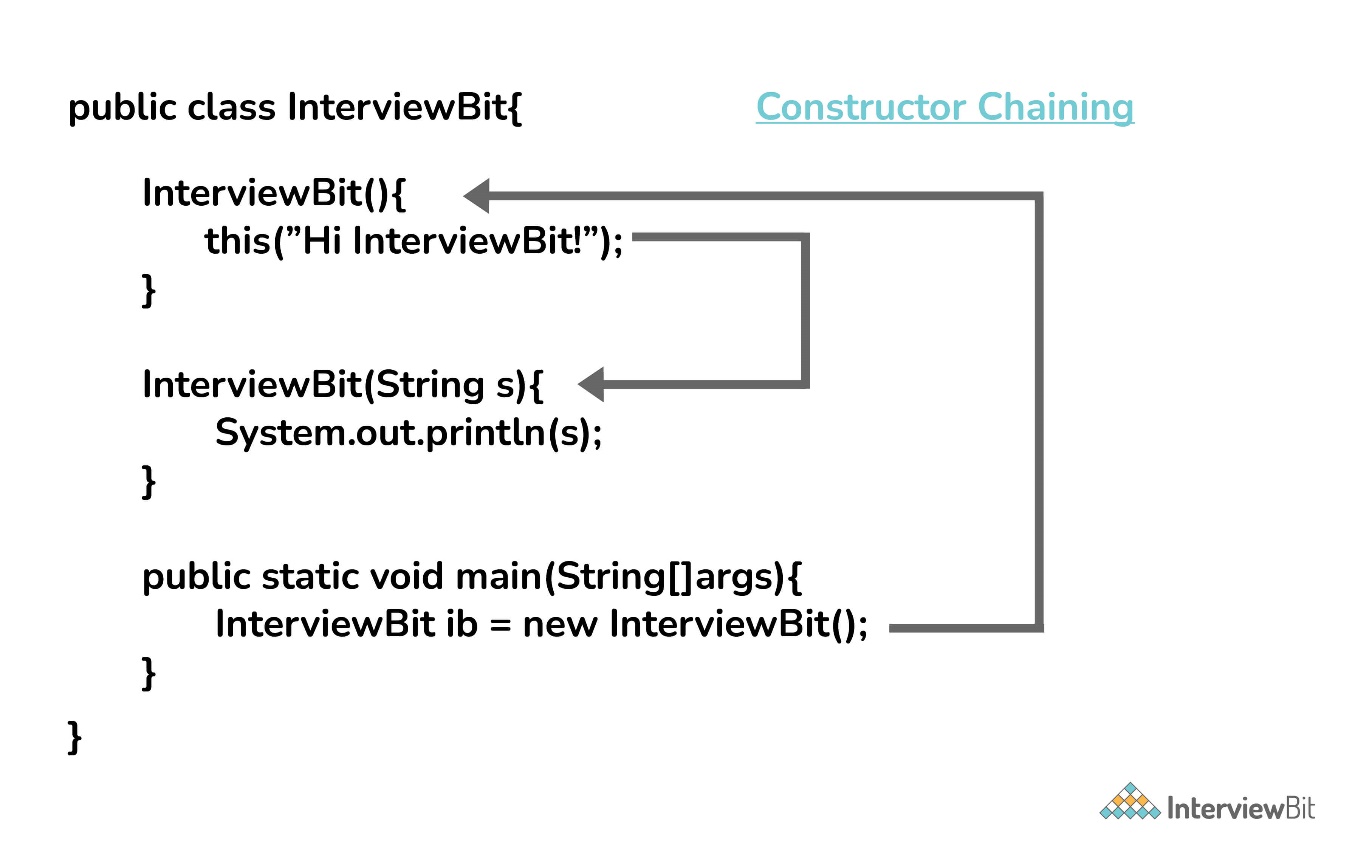
}

}

finally block will be executed irrespective of the exception or not. The only case where finally block is not executed is when it encounters ‘System.exit()’ method anywhere in try/catch block.

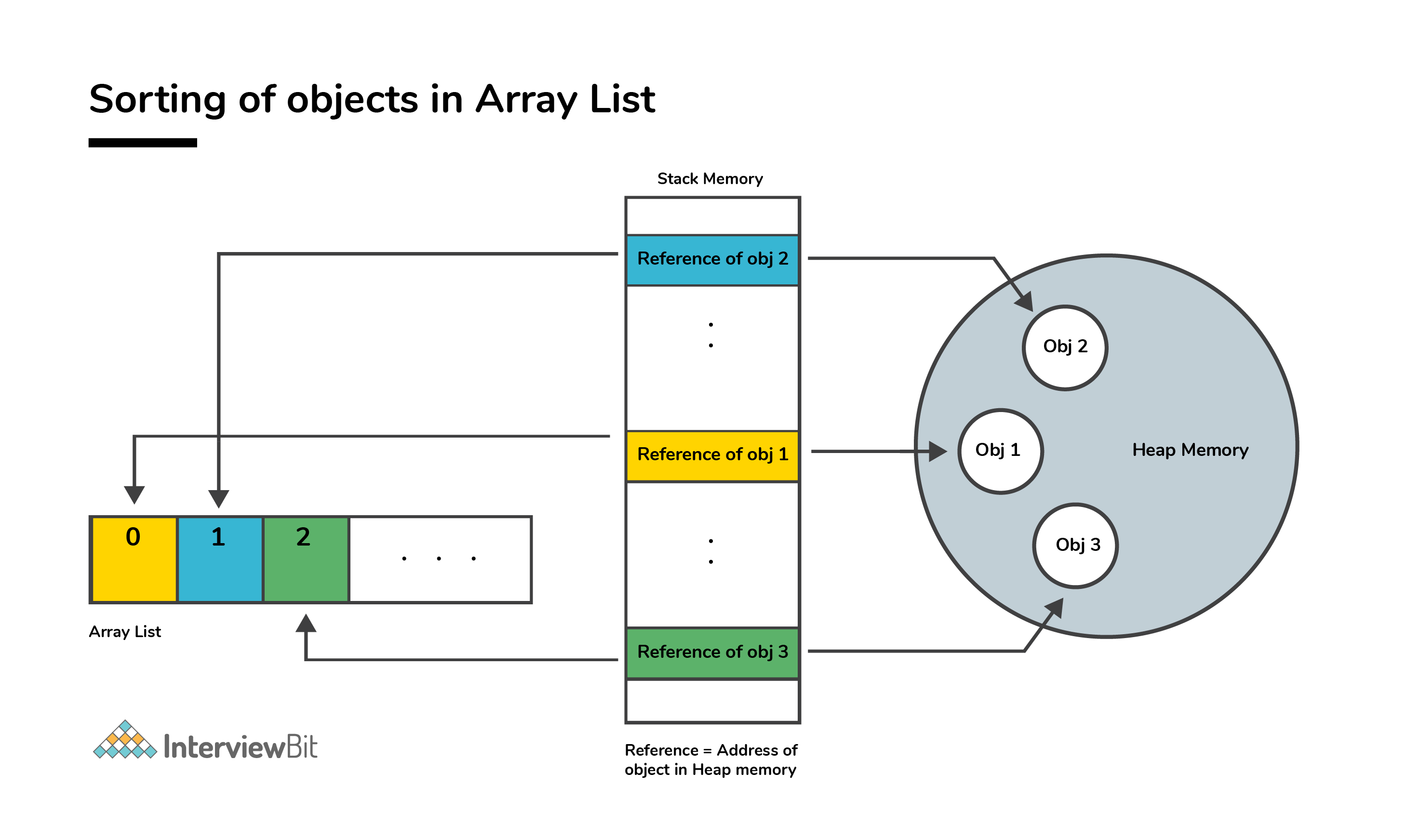
### 39. Can you call a constructor of a class inside the another constructor?

Yes, the concept can be termed as constructor chaining and can be achieved using this().

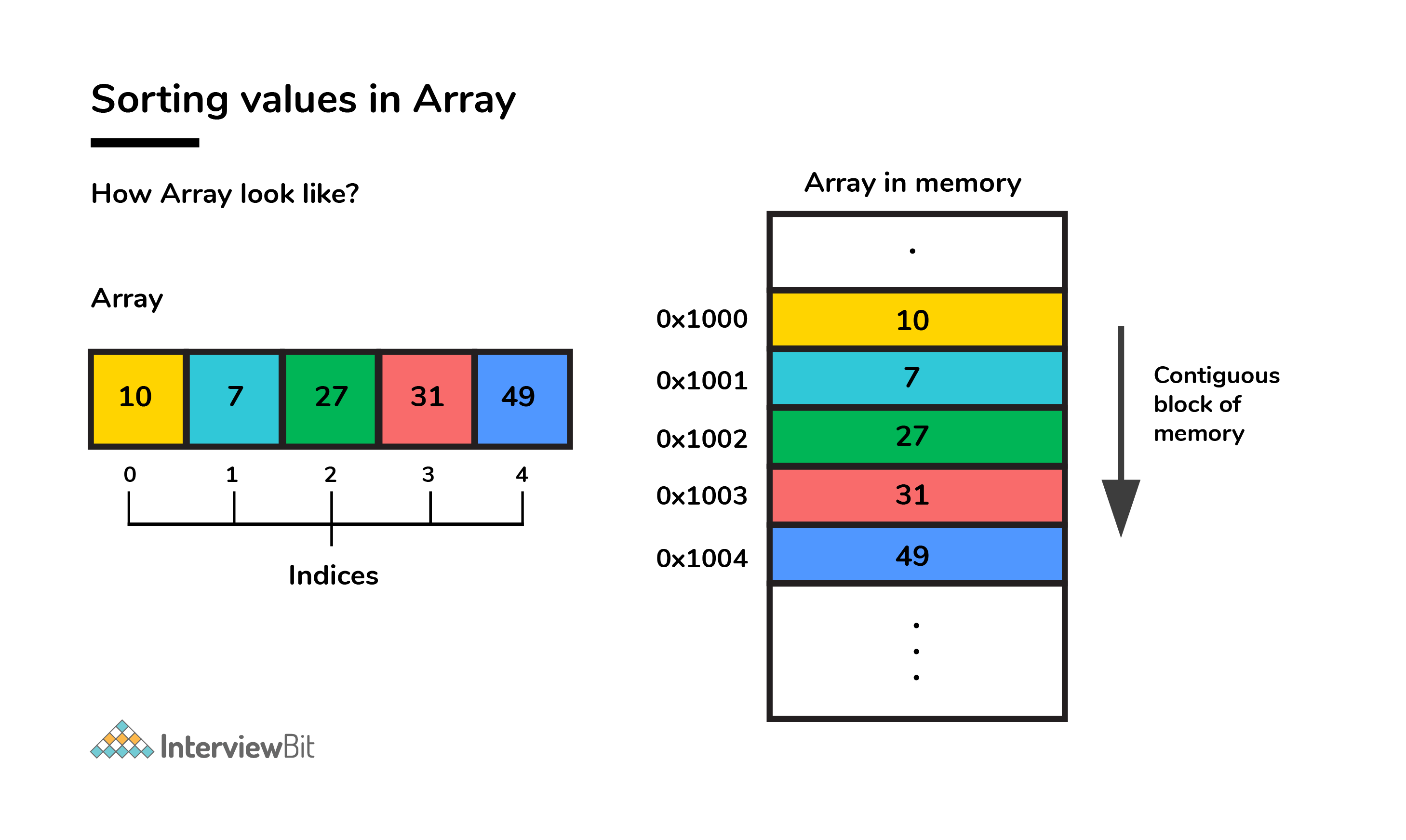


### 40. Contiguous memory locations are usually used for storing actual values in an array but not in ArrayList. Explain.

In the case of ArrayList, data storing in the form of primitive data types (like int, float, etc.) is not possible. The data members/objects present in the ArrayList have references to the objects which are located at various sites in the memory. Thus, storing of actual objects or non-primitive data types (like Integer, Double, etc.) takes place in various memory locations.



However, the same does not apply to the arrays. Object or primitive type values can be stored in arrays in contiguous memory locations, hence every element does not require any reference to the next element.



## Java Advanced Interview Questions

### 41. Although inheritance is a popular OOPs concept, it is less advantageous than composition. Explain.

Inheritance lags behind composition in the following scenarios:

* Multiple-inheritance is not possible in Java. Classes can only extend from one superclass. In cases where multiple functionalities are required, for example - to read and write information into the file, the pattern of composition is preferred. The writer, as well as reader functionalities, can be made use of by considering them as the private members.
* Composition assists in attaining high flexibility and prevents breaking of encapsulation.
* Unit testing is possible with composition and not inheritance. When a developer wants to test a class composing a different class, then Mock Object can be created for signifying the composed class to facilitate testing. This technique is not possible with the help of inheritance as the derived class cannot be tested without the help of the superclass in inheritance.
* The loosely coupled nature of composition is preferable over the tightly coupled nature of inheritance.

Let’s take an example:

package comparison;

public class Top {

public int start() {

return 0;

}

}

class Bottom extends Top {

public int stop() {

return 0;

}

}

In the above example, inheritance is followed. Now, some modifications are done to the Top class like this:

public class Top {

public int start() {

return 0;

}

public void stop() {

}

}

If the new implementation of the Top class is followed, a compile-time error is bound to occur in the Bottom class. Incompatible return type is there for the Top.stop() function. Changes have to be made to either the Top or the Bottom class to ensure compatibility. However, the composition technique can be utilized to solve the given problem:

class Bottom {

Top par = new Top();

public int stop() {

par.start();

par.stop();

return 0;

}

}

### 42. How is the creation of a String using new() different from that of a literal?

When a String is formed as a literal with the assistance of an assignment operator, it makes its way into the String constant pool so that String Interning can take place. This same object in the heap will be referenced by a different String if the content is the same for both of them.

public bool checking() {

String first = "InterviewBit";

String second = "InterviewBit";

if (first == second)

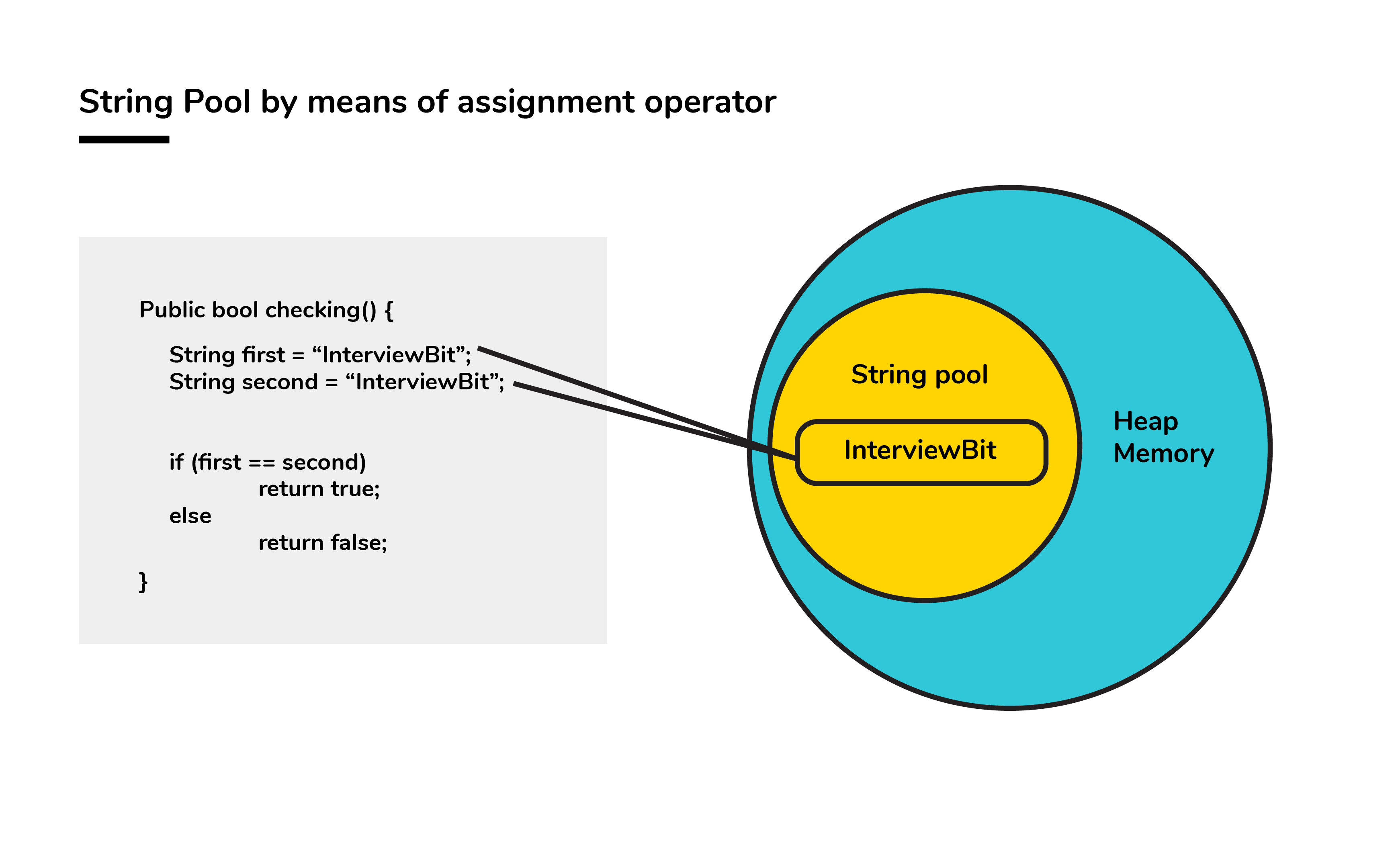
return true;

else

return false;

}

The checking() function will return true as the same content is referenced by both the variables.



Conversely, when a String formation takes place with the help of a new() operator, interning does not take place. The object gets created in the heap memory even if the same content object is present.

public bool checking() {

String first = new String("InterviewBit");

String second = new String("InterviewBit");

if (first == second)

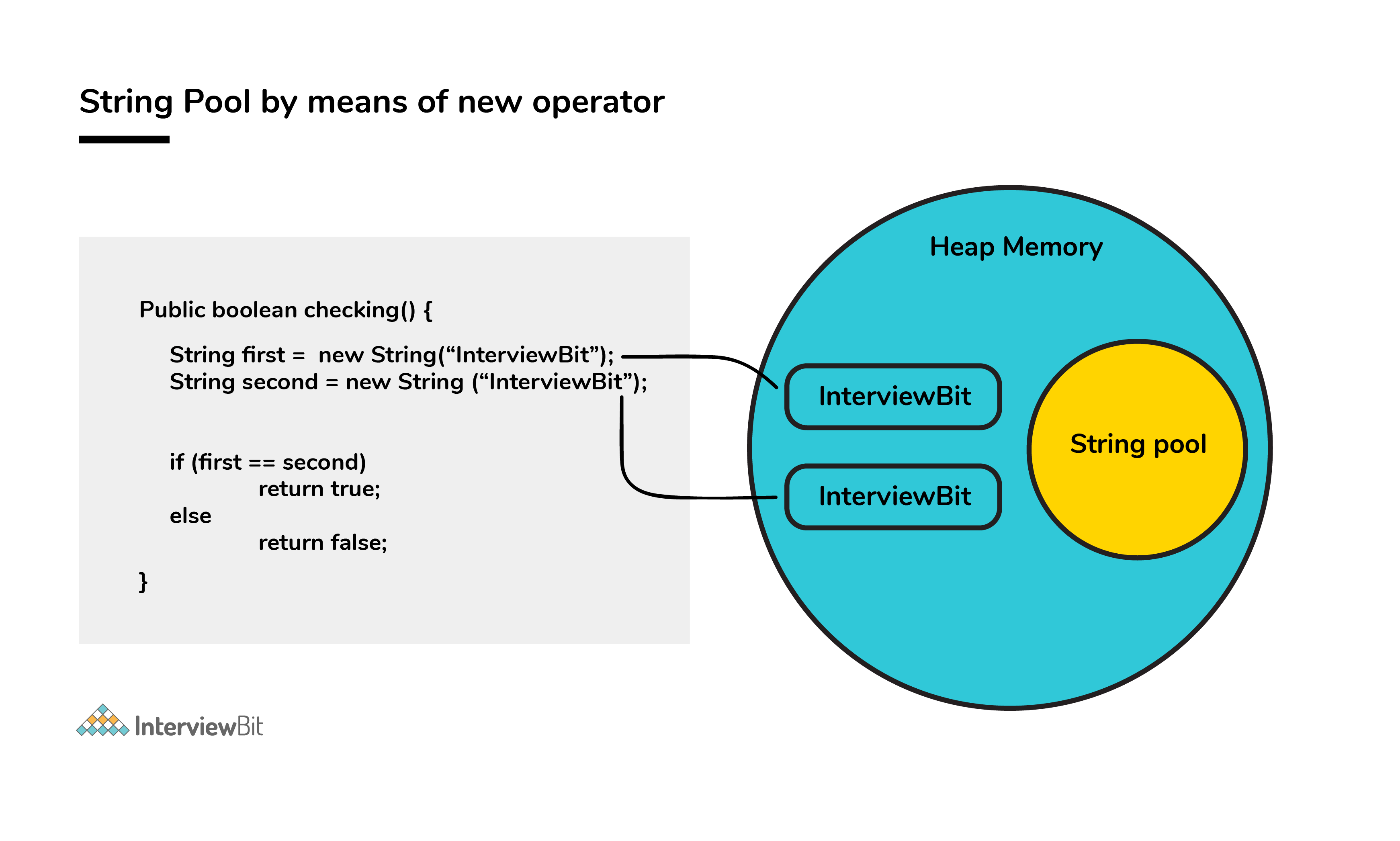
return true;

else

return false;

}

The checking() function will return false as the same content is not referenced by both the variables.



### 43. Is exceeding the memory limit possible in a program despite having a garbage collector?

Yes, it is possible for the program to go out of memory in spite of the presence of a garbage collector. Garbage collection assists in recognizing and eliminating those objects which are not required in the program anymore, in order to free up the resources used by them.

In a program, if an object is unreachable, then the execution of garbage collection takes place with respect to that object. If the amount of memory required for creating a new object is not sufficient, then memory is released for those objects which are no longer in the scope with the help of a garbage collector. The memory limit is exceeded for the program when the memory released is not enough for creating new objects.

Moreover, exhaustion of the heap memory takes place if objects are created in such a manner that they remain in the scope and consume memory. The developer should make sure to dereference the object after its work is accomplished. Although the garbage collector endeavors its level best to reclaim memory as much as possible, memory limits can still be exceeded.

Let’s take a look at the following example:

List<String> example = new LinkedList<String>();

while(true){

example.add(new String("Memory Limit Exceeded"));

}

### 44. Why is synchronization necessary? Explain with the help of a relevant example.

Concurrent execution of different processes is made possible by synchronization. When a particular resource is shared between many threads, situations may arise in which multiple threads require the same shared resource.

Synchronization assists in resolving the issue and the resource is shared by a single thread at a time. Let’s take an example to understand it more clearly. For example, you have a URL and you have to find out the number of requests made to it. Two simultaneous requests can make the count erratic.

**No synchronization:**

package anonymous;

public class Counting {

private int increase\_counter;

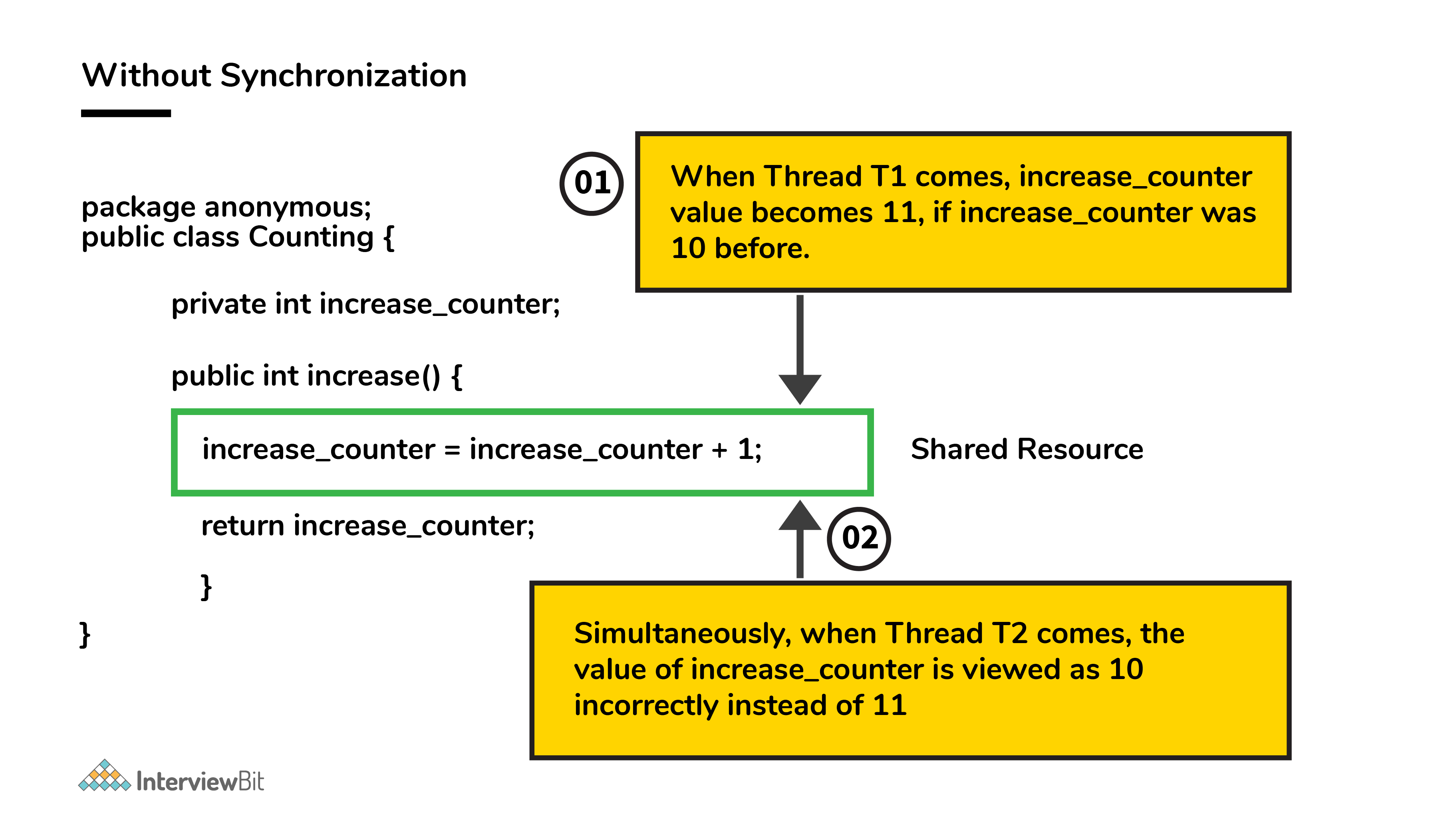
public int increase() {

increase\_counter = increase\_counter + 1;

return increase\_counter;

}

}



If a thread Thread1 views the count as 10, it will be increased by 1 to 11. Simultaneously, if another thread Thread2 views the count as 10, it will be increased by 1 to 11. Thus, inconsistency in count values takes place because the expected final value is 12 but the actual final value we get will be 11.

Now, the function increase() is made synchronized so that simultaneous accessing cannot take place.

**With synchronization:**

package anonymous;

public class Counting {

private int increase\_counter;

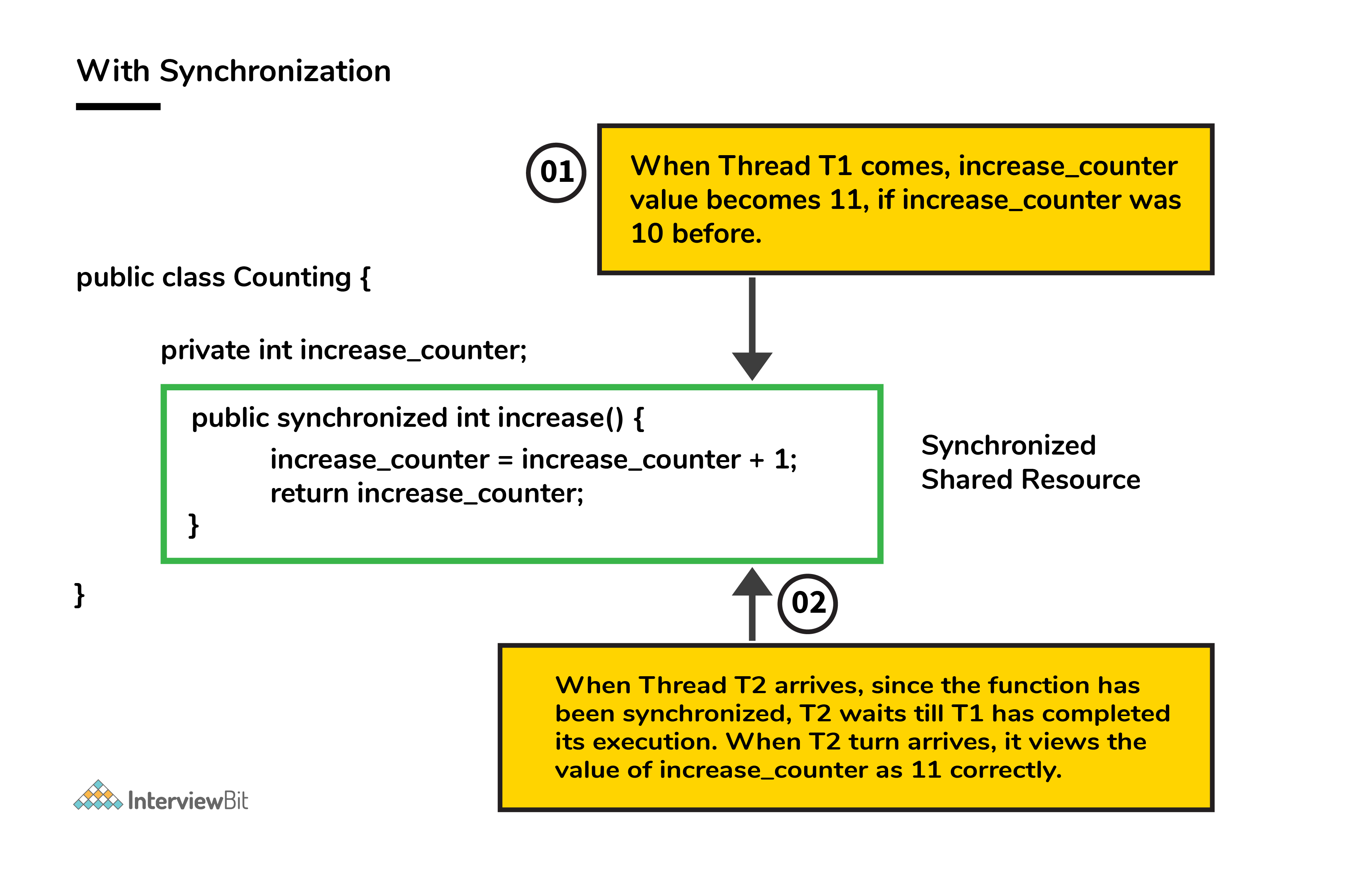
public synchronized int increase() {

increase\_counter = increase\_counter + 1;

return increase\_counter;

}

}



If a thread Thread1 views the count as 10, it will be increased by 1 to 11, then the thread Thread2 will view the count as 11, it will be increased by 1 to 12. Thus, consistency in count values takes place.

### 45. In the given code below, what is the significance of ... ?

public void fooBarMethod(String... variables){

// method code

}

* Ability to provide ... is a feature called varargs (variable arguments) which was introduced as part of Java 5.
* The function having ... in the above example indicates that it can receive multiple arguments of the datatype String.
* For example, the fooBarMethod can be called in multiple ways and we can still have one method to process the data as shown below:

fooBarMethod("foo", "bar");

fooBarMethod("foo", "bar", "boo");

fooBarMethod(new String[]{"foo", "var", "boo"});

public void myMethod(String... variables){

for(String variable : variables){

// business logic

}

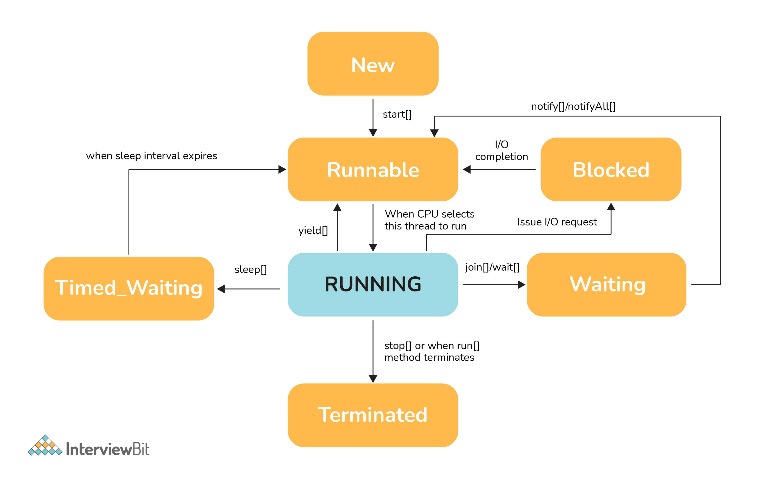
}

### 46. Can you explain the Java thread lifecycle?

Java thread life cycle is as follows:

* **New** – When the instance of the thread is created and the start() method has not been invoked, the thread is considered to be alive and hence in the NEW state.
* **Runnable** – Once the start() method is invoked, before the run() method is called by JVM, the thread is said to be in RUNNABLE (ready to run) state. This state can also be entered from the Waiting or Sleeping state of the thread.
* **Running** – When the run() method has been invoked and the thread starts its execution, the thread is said to be in a RUNNING state.
* **Non-Runnable (Blocked/Waiting)** – When the thread is not able to run despite the fact of its aliveness, the thread is said to be in a NON-RUNNABLE state. Ideally, after some time of its aliveness, the thread should go to a runnable state.
  + A thread is said to be in a Blocked state if it wants to enter synchronized code but it is unable to as another thread is operating in that synchronized block on the same object. The first thread has to wait until the other thread exits the synchronized block.
  + A thread is said to be in a Waiting state if it is waiting for the signal to execute from another thread, i.e it waits for work until the signal is received.
* **Terminated** – Once the run() method execution is completed, the thread is said to enter the TERMINATED step and is considered to not be alive.

The following flowchart clearly explains the lifecycle of the thread in Java.



### 47. What could be the tradeoff between the usage of an unordered array versus the usage of an ordered array?

* The main advantage of having an ordered array is the reduced search time complexity of O(log n) whereas the time complexity in an unordered array is O(n).
* The main drawback of the ordered array is its increased insertion time which is O(n) due to the fact that its element has to reordered to maintain the order of array during every insertion whereas the time complexity in the unordered array is only O(1).
* Considering the above 2 key points and depending on what kind of scenario a developer requires, the appropriate data structure can be used for implementation.

### 48. Is it possible to import the same class or package twice in Java and what happens to it during runtime?

It is possible to import a class or package more than once, however, it is redundant because the JVM internally loads the package or class only once.

### 49. In case a package has sub packages, will it suffice to import only the main package? e.g. Does importing of com.myMainPackage.\* also import com.myMainPackage.mySubPackage.\*?

This is a big NO. We need to understand that the importing of the sub-packages of a package needs to be done explicitly. Importing the parent package only results in the import of the classes within it and not the contents of its child/sub-packages.

### 50. Will the finally block be executed if the code System.exit(0) is written at the end of try block?

NO. The control of the program post System.exit(0) is immediately gone and the program gets terminated which is why the finally block never gets executed.

### 51. What do you understand by marker interfaces in Java?

Marker interfaces, also known as tagging interfaces are those interfaces that have no methods and constants defined in them. They are there for helping the compiler and JVM to get run time-related information regarding the objects.

### 52. Explain the term “Double Brace Initialisation” in Java?

This is a convenient means of initializing any collections in Java. Consider the below example.

import java.util.HashSet;

import java.util.Set;

public class IBDoubleBraceDemo{

public static void main(String[] args){

Set<String> stringSets = new HashSet<String>()

{

{

add("set1");

add("set2");

add("set3");

}

};

doSomething(stringSets);

}

private static void doSomething(Set<String> stringSets){

System.out.println(stringSets);

}

}

In the above example, we see that the stringSets were initialized by using double braces.

* The first brace does the task of creating an anonymous inner class that has the capability of accessing the parent class’s behavior. In our example, we are creating the subclass of HashSet so that it can use the add() method of HashSet.
* The second braces do the task of initializing the instances.

Care should be taken while initializing through this method as the method involves the creation of anonymous inner classes which can cause problems during the garbage collection or serialization processes and may also result in memory leaks.

### 53. Why is it said that the length() method of String class doesn't return accurate results?

* The length method returns the number of Unicode units of the String. Let's understand what Unicode units are and what is the confusion below.
* We know that Java uses UTF-16 for String representation. With this Unicode, we need to understand the below two Unicode related terms:
  + Code Point: This represents an integer denoting a character in the code space.
  + Code Unit: This is a bit sequence used for encoding the code points. In order to do this, one or more units might be required for representing a code point.
* Under the UTF-16 scheme, the code points were divided logically into 17 planes and the first plane was called the Basic Multilingual Plane (BMP). The BMP has classic characters - U+0000 to U+FFFF. The rest of the characters- U+10000 to U+10FFFF were termed as the supplementary characters as they were contained in the remaining planes.
  + The code points from the first plane are encoded using **one** 16-bit code unit
  + The code points from the remaining planes are encoded using **two** code units.

Now if a string contained supplementary characters, the length function would count that as 2 units and the result of the length() function would not be as per what is expected.

In other words, if there is 1 supplementary character of 2 units, the length of that SINGLE character is considered to be TWO - Notice the inaccuracy here? As per the java documentation, it is expected, but as per the real logic, it is inaccurate.

### 54. What is the output of the below code and why?

public class InterviewBit{

public static void main(String[] args)

{

System.out.println('b' + 'i' + 't');

}

}

“bit” would have been the result printed if the letters were used in double-quotes (or the string literals). But the question has the character literals (single quotes) being used which is why concatenation wouldn't occur. The corresponding ASCII values of each character would be added and the result of that sum would be printed.  
The ASCII values of ‘b’, ‘i’, ‘t’ are:

* ‘b’ = 98
* ‘i’ = 105
* ‘t’ = 116

98 + 105 + 116 = 319

Hence 319 would be printed.

### 55. What are the possible ways of making object eligible for garbage collection (GC) in Java?

**First Approach:** Set the object references to null once the object creation purpose is served.

public class IBGarbageCollect {

public static void main (String [] args){

String s1 = "Some String";

// s1 referencing String object - not yet eligible for GC

s1 = null; // now s1 is eligible for GC

}

}

**Second Approach:** Point the reference variable to another object. Doing this, the object which the reference variable was referencing before becomes eligible for GC.

public class IBGarbageCollect {

public static void main(String [] args){

String s1 = "To Garbage Collect";

String s2 = "Another Object";

System.out.println(s1); // s1 is not yet eligible for GC

s1 = s2; // Point s1 to other object pointed by s2

/\* Here, the string object having the content "To Garbage Collect" is not referred by any reference variable. Therefore, it is eligible for GC \*/

}

}

**Third Approach:** Island of Isolation Approach: When 2 reference variables pointing to instances of the same class, and these variables refer to only each other and the objects pointed by these 2 variables don't have any other references, then it is said to have formed an “Island of Isolation” and these 2 objects are eligible for GC.

public class IBGarbageCollect {

IBGarbageCollect ib;

public static void main(String [] str){

IBGarbageCollect ibgc1 = new IBGarbageCollect();

IBGarbageCollect ibgc2 = new IBGarbageCollect();

ibgc1.ib = ibgc2; //ibgc1 points to ibgc2

ibgc2.ib = ibgc1; //ibgc2 points to ibgc1

ibgc1 = null;

ibgc2 = null;

/\*

\* We see that ibgc1 and ibgc2 objects refer

\* to only each other and have no valid

\* references- these 2 objects for island of isolcation - eligible for GC

\*/

}

}

## Java Interview Programs

### 56. Check if a given string is palindrome using recursion.

/\*

\* Java program to check if a given inputted string is palindrome or not using recursion.

\*/

import java.util.\*;

public class InterviewBit {

public static void main(String args[]) {

Scanner s = new Scanner(System.in);

String word = s.nextLine();

System.out.println("Is "+word+" palindrome? - "+isWordPalindrome(word));

}

public static boolean isWordPalindrome(String word){

String reverseWord = getReverseWord(word);

//if word equals its reverse, then it is a palindrome

if(word.equals(reverseWord)){

return true;

}

return false;

}

public static String getReverseWord(String word){

if(word == null || word.isEmpty()){

return word;

}

return word.charAt(word.length()- 1) + getReverseWord(word.substring(0, word.length() - 1));

}

}

### 57. Write a Java program to check if the two strings are anagrams.

The main idea is to validate the length of strings and then if found equal, convert the string to char array and then sort the arrays and check if both are equal.

import java.util.Arrays;

import java.util.Scanner;

public class InterviewBit {

public static void main(String[] args) {

Scanner s = new Scanner(System.in);

//Input from two strings

System.out.print("First String: ");

String string1 = s.nextLine();

System.out.print("Second String: ");

String string2 = s.nextLine();

// check for the length

if(string1.length() == string2.length()) {

// convert strings to char array

char[] characterArray1 = string1.toCharArray();

char[] characterArray2 = string2.toCharArray();

// sort the arrays

Arrays.sort(characterArray1);

Arrays.sort(characterArray2);

// check for equality, if found equal then anagram, else not an anagram

boolean isAnagram = Arrays.equals(characterArray1, characterArray2);

System.out.println("Anagram: "+ isAnagram);

}

}

### 58. Write a Java Program to find the factorial of a given number.

public class FindFactorial {

public static void main(String[] args) {

int num = 10;

long factorialResult = 1l;

for(int i = 1; i <= num; ++i)

{

factorialResult \*= i;

}

System.out.println("Factorial: "+factorialResult);

}

}

### 59. Given an array of non-duplicating numbers from 1 to n where one number is missing, write an efficient java program to find that missing number.

Idea is to find the sum of n natural numbers using the formula and then finding the sum of numbers in the given array. Subtracting these two sums results in the number that is the actual missing number. This results in O(n) time complexity and O(1) space complexity.

public class IBMissingNumberProblem {

public static void main(String[] args) {

int[] array={4,3,8,7,5,2,6};

int missingNumber = findMissingNum(array);

System.out.println("Missing Number is "+ missingNumber);

}

public static int findMissingNum(int[] array) {

int n=array.length+1;

int sumOfFirstNNums=n\*(n+1)/2;

int actualSumOfArr=0;

for (int i = 0; i < array.length; i++) {

actualSumOfArr+=array[i];

}

return sumOfFirstNNums-actualSumOfArr;

}

}

### 60. Write a Java Program to check if any number is a magic number or not. A number is said to be a magic number if after doing sum of digits in each step and inturn doing sum of digits of that sum, the ultimate result (when there is only one digit left) is 1.

Example, consider the number:

* Step 1: 163 => 1+6+3 = 10
* Step 2: 10 => 1+0 = 1 => Hence 163 is a magic number

public class IBMagicNumber{

public static void main(String[] args) {

int num = 163;

int sumOfDigits = 0;

while (num > 0 || sumOfDigits > 9)

{

if (num == 0)

{

num = sumOfDigits;

sumOfDigits = 0;

}

sumOfDigits += num % 10;

num /= 10;

}

// If sum is 1, original number is magic number

if(sumOfDigits == 1) {

System.out.println("Magic number");

}else {

System.out.println("Not magic number");

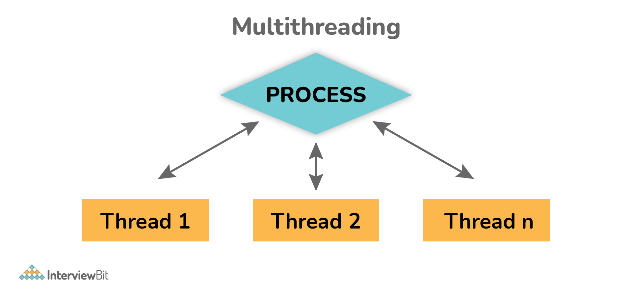
}

}

}

### What do you mean by Multithreading? Why is it important?

Multithreading means multiple threads and is considered one of the most important [features of Java](https://www.interviewbit.com/java-interview-questions/). As the name suggests, it is the ability of a CPU to execute multiple threads independently at the same time but sharing the process resources simultaneously. Its main purpose is to provide simultaneous execution of multiple threads to utilize the CPU time as much as possible. It is a Java feature where one can subdivide the specific program into two or more threads to make the execution of the program fast and easy.



Crack your next tech interview with confidence!

Take a free mock interview, get instant⚡️ feedback and recommendation💡

[Take Free Mock Interview](https://www.interviewbit.com/event/free-mock-coding-interview?utm_source=ib&utm_medium=in-interview-guide&utm_campaign=MoCo&utm_content=/multithreading-interview-questions/)

## Multithreading Interview Questions in Java for Freshers

### 1. What are the benefits of using Multithreading?

There are various benefits of multithreading as given below:

* Allow the program to run continuously even if a part of it is blocked.
* Improve performance as compared to traditional parallel programs that use multiple processes.
* Allows to write effective programs that utilize maximum CPU time
* Improves the responsiveness of complex applications or programs.
* Increase use of CPU resources and reduce costs of maintenance.
* Saves time and parallelism tasks.
* If an exception occurs in a single thread, it will not affect other threads as threads are independent.
* Less resource-intensive than executing multiple processes at the same time.

### 2. What is Thread in Java?

Threads are basically the lightweight and smallest unit of processing that can be managed independently by a scheduler. Threads are referred to as parts of a process that simply let a program execute efficiently with other parts or threads of the process at the same time. Using threads, one can perform complicated tasks in the easiest way. It is considered the simplest way to take advantage of multiple CPUs available in a machine. They share the common address space and are independent of each other.

### 3. What are the two ways of implementing thread in Java?

There are basically two ways of implementing thread in java as given below:

* Implementing **Runnable** interface in Java

Example:

class MultithreadingDemo extends Thread

{

public void run()

{

System.out.println("My thread is in running state.");

}

public static void main(String args[])

{

MultithreadingDemoobj=new MultithreadingDemo();

obj.start();

}

}

Output:

My thread is in running state.

* Extending the **Thread** class.

Example:

class MultithreadingDemo implements Runnable

{

public void run()

{

System.out.println("My thread is in running state.");

}

public static void main(String args[])

{

MultithreadingDemo obj=new MultithreadingDemo();

Threadtobj =new Thread(obj); tobj.start();

}

}

Output:

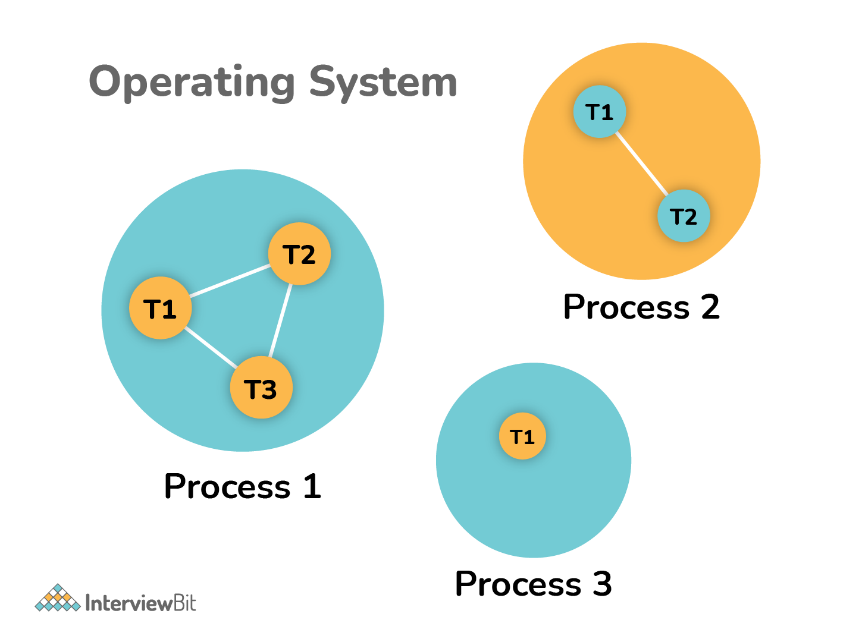
My thread is in running state.

You can download a PDF version of Multithreading Interview Questions.

[Download PDF](javascript:void(0))

### 4. What's the difference between thread and process?

**Thread**: It simply refers to the smallest units of the particular process. It has the ability to execute different parts (referred to as thread) of the program at the same time.    
  
**Process**: It simply refers to a program that is in execution i.e., an active program. A process can be handled using PCB (Process Control Block).



| **Thread** | **Process** |
| --- | --- |
| It is a subset of a subunit of a process. | It is a program in execution containing multiple threads. |
| In this, inter-thread communication is faster, less expensive, easy and efficient because threads share the same memory address of the process they belong to. | In this, inter-process communication is slower, expensive, and complex because each process has different memory space or address., |
| These are easier to create, lightweight, and have less overhead. | These are difficult to create, heavyweight, and have more overhead. |
| It requires less time for creation, termination, and context switching. | It requires more time for creation, termination, and context switching. |
| Processes with multiple threads use fewer resources. | Processes without threads use more resources. |
| Threads are parts of a process, so they are dependent on each other but each thread executes independently. | Processes are independent of each other. |
| There is a need for synchronization in threads to avoid unexpected scenarios or problems. | There is no need for synchronization in each process. |
| They share data and information with each other. | They do not share data with each other. |

### 5. What’s the difference between class lock and object lock?

**Class Lock**: In java, each and every class has a unique lock usually referred to as a class level lock. These locks are achieved using the keyword ‘static synchronized’ and can be used to make static data thread-safe. It is generally used when one wants to prevent multiple threads from entering a synchronized block.   
  
Example:

public class ClassLevelLockExample

{

public void classLevelLockMethod()

{

synchronized (ClassLevelLockExample.class)

{

//DO your stuff here

}

}

}

**Object Lock**: In java, each and every object has a unique lock usually referred to as an object-level lock. These locks are achieved using the keyword ‘synchronized’ and can be used to protect non-static data. It is generally used when one wants to synchronize a non-static method or block so that only the thread will be able to execute the code block on a given instance of the class.    
  
Example:

public class ObjectLevelLockExample

{

public void objectLevelLockMethod()

{

synchronized (this)

{

//DO your stuff here

}

}

}

### 6. What's the difference between User thread and Daemon thread?

User and Daemon are basically two types of thread used in Java by using a ‘Thread Class’.    
  
**User Thread (Non-Daemon Thread)**: In Java, user threads have a specific life cycle and its life is independent of any other thread. JVM (Java Virtual Machine) waits for any of the user threads to complete its tasks before terminating it. When user threads are finished, JVM terminates the whole program along with associated daemon threads.   
  
**Daemon Thread**: In Java, daemon threads are basically referred to as a service provider that provides services and support to user threads. There are basically two methods available in thread class for daemon thread: setDaemon() and isDaemon().   
  
**User Thread vs Daemon Thread**

| **User Thread** | **Daemon Thread** |
| --- | --- |
| JVM waits for user threads to finish their tasks before termination. | JVM does not wait for daemon threads to finish their tasks before termination. |
| These threads are normally created by the user for executing tasks concurrently. | These threads are normally created by JVM. |
| They are used for critical tasks or core work of an application. | They are not used for any critical tasks but to do some supporting tasks. |
| These threads are referred to as high-priority tasks, therefore are required for running in the foreground. | These threads are referred to as low priority threads, therefore are especially required for supporting background tasks like garbage collection, releasing memory of unused objects, etc. |

### 7. How can we create daemon threads?

We can create daemon threads in java using the thread class **setDaemon(true)**. It is used to mark the current thread as daemon thread or user thread. **isDaemon()** method is generally used to check whether the current thread is daemon or not. If the thread is a daemon, it will return true otherwise it returns false.    
Example:     
**Program to illustrate the use of setDaemon() and isDaemon() method.**

public class DaemonThread extends Thread

{

public DaemonThread(String name){

super(name);

}

public void run()

{

// Checking whether the thread is Daemon or not

if(Thread.currentThread().isDaemon())

{

System.out.println(getName() + " is Daemon thread");

}

else

{

System.out.println(getName() + " is User thread");

}

}

public static void main(String[] args)

{

DaemonThread t1 = new DaemonThread("t1");

DaemonThread t2 = new DaemonThread("t2");

DaemonThread t3 = new DaemonThread("t3");

// Setting user thread t1 to Daemon

t1.setDaemon(true);

// starting first 2 threads

t1.start();

t2.start();

// Setting user thread t3 to Daemon

t3.setDaemon(true);

t3.start();

}

}

Output:

t1 is Daemon thread

t3 is Daemon thread

t2 is User thread

But one can only call the **setDaemon()** method before start() method otherwise it will definitely throw IllegalThreadStateException as shown below:

public class DaemonThread extends Thread

{

public void run()

{

System.out.println("Thread name: " + Thread.currentThread().getName());

System.out.println("Check if its DaemonThread: "

+ Thread.currentThread().isDaemon());

}

public static void main(String[] args)

{

DaemonThread t1 = new DaemonThread();

DaemonThread t2 = new DaemonThread();

t1.start();

// Exception as the thread is already started

t1.setDaemon(true);

t2.start();

}

}

Output:

Thread name: Thread-0

Check if its DaemonThread: false

### 8. What are the wait() and sleep() methods?

**wait()**: As the name suggests, it is a non-static method that causes the current thread to wait and go to sleep until some other threads call the notify () or notifyAll() method for the object’s monitor (lock). It simply releases the lock and is mostly used for inter-thread communication. It is defined in the object class, and should only be called from a synchronized context.

Example:

synchronized(monitor)

{

monitor.wait(); Here Lock Is Released by Current Thread

}

**sleep()**: As the name suggests, it is a static method that pauses or stops the execution of the current thread for some specified period. It doesn’t release the lock while waiting and is mostly used to introduce pause on execution. It is defined in thread class, and no need to call from a synchronized context.

Example:

synchronized(monitor)

{

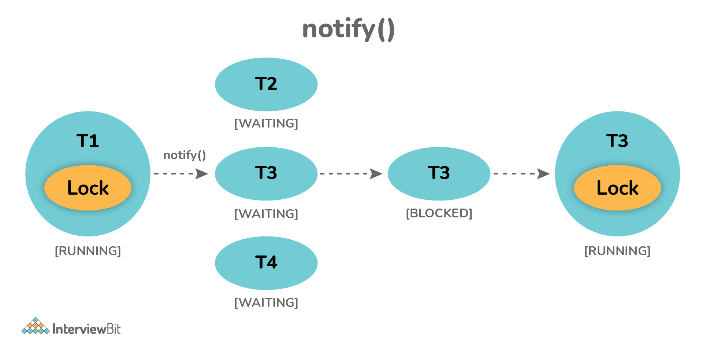
Thread.sleep(1000); Here Lock Is Held by The Current Thread

//after 1000 milliseconds, the current thread will wake up, or after we call that is interrupt() method

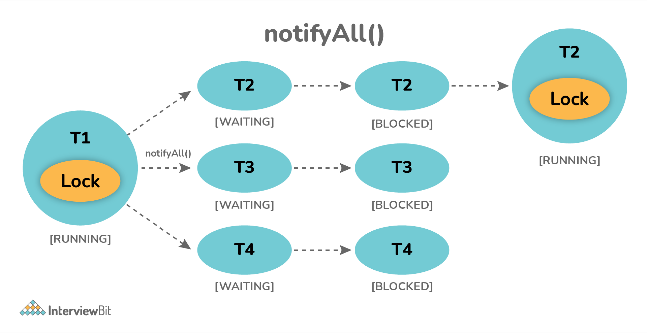
}

### 9. What’s the difference between notify() and notifyAll()?

**notify()**: It sends a notification and wakes up only a single thread instead of multiple threads that are waiting on the object’s monitor.



**notifyAll()**: It sends notifications and wakes up all threads and allows them to compete for the object's monitor instead of a single thread.



### 10. Why wait(), notify(), and notifyAll() methods are present in Object class?

We know that every object has a monitor that allows the thread to hold a lock on the object. But the thread class doesn't contain any monitors. Thread usually waits for the object’s monitor (lock) by calling the wait() method on an object, and notify other threads that are waiting for the same lock using notify() or notifyAll() method.  Therefore, these three methods are called on objects only and allow all threads to communicate with each that are created on that object.

### 11. What is Runnable and Callable Interface? Write the difference between them.

Both the interfaces are generally used to encapsulate tasks that are needed to be executed by another thread. But there are some differences between them as given below:   
  
**Running Interface**: This interface is basically available in Java right from the beginning. It is simply used to execute code on a concurrent thread.    
**Callable Interface**: This interface is basically a new one that was introduced as a part of the concurrency package. It addresses the limitation of runnable interfaces along with some major changes like generics, enum, static imports, variable argument method, etc. It uses generics to define the return type of object.

public interface Runnable

{

public abstract void run();

}

public interface Callable<V>

{

V call() throws Exception;

}

**Runnable Interface vs Callable Interface**

| **Runnable Interface** | **Callable Interface** |
| --- | --- |
| It does not return any result and therefore, cannot throw a checked exception. | It returns a result and therefore, can throw an exception. |
| It cannot be passed to invokeAll method. | It can be passed to invokeAll method. |
| It was introduced in JDK 1.0. | It was introduced in JDK 5.0, so one cannot use it before Java 5. |
| It simply belongs to Java.lang. | It simply belongs to java.util.concurrent. |
| It uses the run() method to define a task. | It uses the call() method to define a task. |
| To use this interface, one needs to override the run() method. | To use this interface, one needs to override the call() method. |

### 12. What is the start() and run() method of Thread class?

**start()**: In simple words, the start() method is used to start or begin the execution of a newly created thread. When the start() method is called, a new thread is created and this newly created thread executes the task that is kept in the run() method. One can call the start() method only once.    
  
**run()**: In simple words, the run() method is used to start or begin the execution of the same thread. When the run() method is called, no new thread is created as in the case of the start() method. This method is executed by the current thread. One can call the run() method multiple times.

### 13. Explain thread pool?

A Thread pool is simply a collection of pre-initialized or worker threads at the start-up that can be used to execute tasks and put back in the pool when completed. It is referred to as pool threads in which a group of fixed-size threads is created.  By reducing the number of application threads and managing their lifecycle, one can mitigate the issue of performance using a thread pool. Using threads, performance can be enhanced and better system stability can occur. To create the thread pools, java.util.concurrent.Executors class usually provides factory methods.

### 14. What’s the purpose of the join() method?

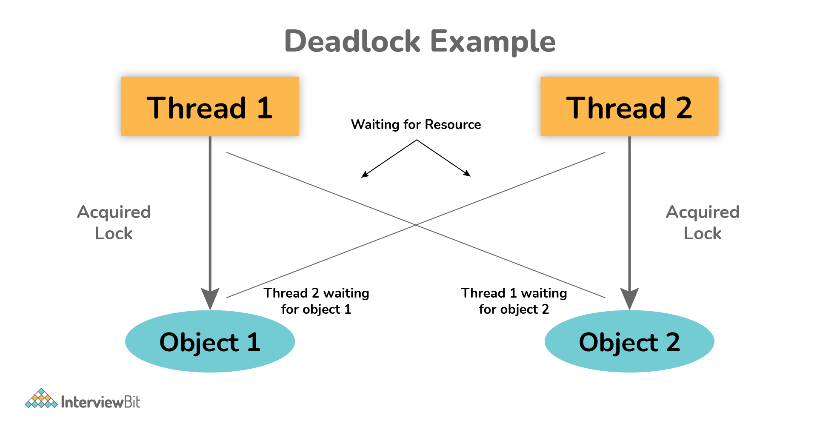
**join()** method is generally used to pause the execution of a current thread unless and until the specified thread on which join is called is dead or completed. To stop a thread from running until another thread gets ended, this method can be used. It joins the start of a thread execution to the end of another thread’s execution. It is considered the final method of a thread class.

### 15. What do you mean by garbage collection?

Garbage collection is basically a process of managing memory automatically. It uses several GC algorithms among which the popular one includes Mark and Sweep. The process includes three phases i.e., marking, deletion, and compaction/copying. In simple words, a garbage collector finds objects that are no longer required by the program and then delete or remove these unused objects to free up the memory space.

### 16. Explain the meaning of the deadlock and when it can occur?

Deadlock, as the name suggests, is a situation where multiple threads are blocked forever. It generally occurs when multiple threads hold locks on different resources and are waiting for other resources to complete their task.



The above diagram shows a deadlock situation where two threads are blocked forever.  Thread 1 is holding Object 1 but needs object 2 to complete processing whereas Thread 2 is holding Object 2 but needs object 1 first. In such conditions, both of them will hold lock forever and will never complete tasks.

### 17. Explain volatile variables in Java?

A volatile variable is basically a keyword that is used to ensure and address the visibility of changes to variables in multithreaded programming. This keyword cannot be used with classes and methods, instead can be used with variables. It is simply used to achieve thread-safety. If you mark any variable as volatile, then all the threads can read its value directly from the main memory rather than CPU cache, so that each thread can get an updated value of the variable.

### 18. How do threads communicate with each other?

Threads can communicate using three methods i.e., wait(), notify(), and notifyAll().

### 19. Can two threads execute two methods (static and non-static concurrently)?

Yes, it is possible. If both the threads acquire locks on different objects, then they can execute concurrently without any problem.

### 20. What is the purpose of the finalize() method?

Finalize() method is basically a method of Object class specially used to perform cleanup operations on unmanaged resources just before garbage collection. It is not at all intended to be called a normal method. After the complete execution of finalize() method, the object gets destroyed automatically.

## Multithreading Interview Questions in Java for Experienced

### 21. What is the synchronization process? Why use it?

Synchronization is basically a process in java that enables a simple strategy for avoiding thread interference and memory consistency errors. This process makes sure that resource will be only used one thread at a time when one thread tries to access a shared resource. It can be achieved in three different ways as given below:

* By the synchronized method
* By synchronized block
* By static synchronization

Syntax:

synchronized (object)

{

//statement to be synchronized

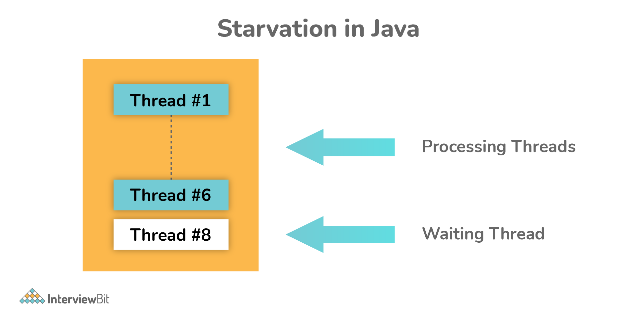
}

### 22. What is synchronized method and synchronized block? Which one should be preferred?

**Synchronized Method**: In this method, the thread acquires a lock on the object when they enter the synchronized method and releases the lock either normally or by throwing an exception when they leave the method.  No other thread can use the whole method unless and until the current thread finishes its execution and release the lock. It can be used when one wants to lock on the entire functionality of a particular method.   
  
**Synchronized Block**: In this method, the thread acquires a lock on the object between parentheses after the synchronized keyword, and releases the lock when they leave the block. No other thread can acquire a lock on the locked object unless and until the synchronized block exists. It can be used when one wants to keep other parts of the programs accessible to other threads.  
   
Synchronized blocks should be preferred more as it boosts the performance of a particular program. It only locks a certain part of the program (critical section) rather than the entire method and therefore leads to less contention.

### 23. What is thread starvation?

Thread starvation is basically a situation or condition where a thread won’t be able to have regular access to shared resources and therefore is unable to proceed or make progress. This is because other threads have high priority and occupy the resources for too long. This usually happens with low-priority threads that do not get CPU for its execution to carry on.

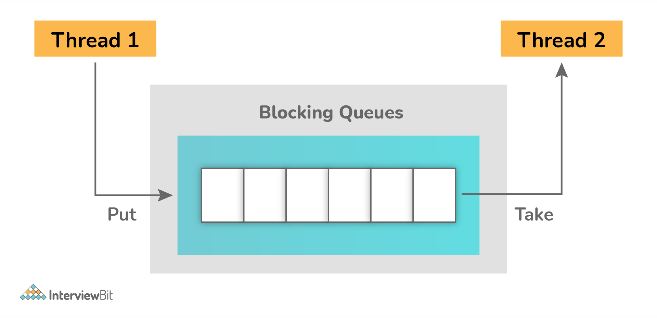


### 24. What is Livelock? What happens when it occurs?

Similar to deadlock, livelock is also another concurrency problem. In this case, the state of threads changes between one another without making any progress. Threads are not blocked but their execution is stopped due to the unavailability of resources.

### 25. What is BlockingQueue?

BlockingQueue basically represents a queue that is thread-safe. Producer thread inserts resource/element into the queue using put() method unless it gets full and consumer thread takes resources from the queue using take() method until it gets empty. But if a thread tries to dequeue from an empty queue, then a particular thread will be blocked until some other thread inserts an item into the queue, or if a thread tries to insert an item into a queue that is already full, then a particular thread will be blocked until some threads take away an item from the queue.



**Example**:

package org.arpit.java2blog;

import java.util.concurrent.ArrayBlockingQueue;

import java.util.concurrent.BlockingQueue;

public class BlockingQueuePCExample {

public static void main(String[] args) {

BlockingQueue<String> queue=new ArrayBlockingQueue<>(5);

Producer producer=new Producer(queue);

Consumer consumer=new Consumer(queue);

Thread producerThread = new Thread(producer);

Thread consumerThread = new Thread(consumer);

producerThread.start();

consumerThread.start();

}

static class Producer implements Runnable {

BlockingQueue<String> queue=null;

public Producer(BlockingQueue queue) {

super();

this.queue = queue;

}

@Override

public void run() {

try {

System.out.println("Producing element 1");

queue.put("Element 1");

Thread.sleep(1000);

System.out.println("Producing element 2");

queue.put("Element 2");

Thread.sleep(1000);

System.out.println("Producing element 3");

queue.put("Element 3");

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

static class Consumer implements Runnable {

BlockingQueue<String> queue=null;

public Consumer(BlockingQueue queue) {

super();

this.queue = queue;

}

@Override

public void run() {

while(true)

{

try {

System.out.println("Consumed "+queue.take());

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

}

**Output**:

Producing element 1

Consumed Element 1

Producing element 2

Consumed Element 2

Producing element 3

Consumed Element 3

### 26. Can you start a thread twice?

No, it's not at all possible to restart a thread once a thread gets started and completes its execution. Thread only runs once and if you try to run it for a second time, then it will throw a runtime exception i.e., java.lang.IllegalThreadStateException.   
  
**Example**:

public class TestThreadTwice1 extends Thread{

public void run(){

System.out.println(" thread is executing now........");

}

public static void main(String args[]){

TestThreadTwice1 t1=new TestThreadTwice1();

t1.start();

t1.start();

}

}

**Output:**

thread is executing now........

Exception in thread "main" java.lang.IllegalThreadStateException

### 27. Explain context switching.

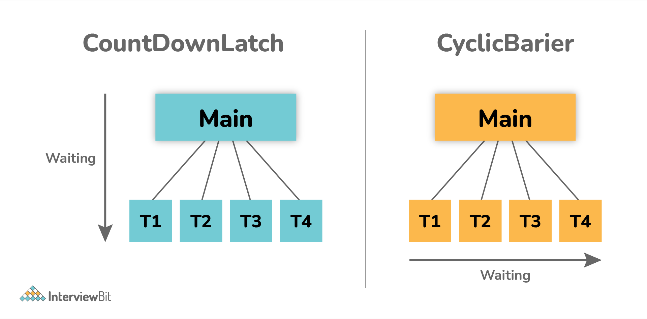
Context switching is basically an important feature of multithreading. It is referred to as switching of CPU from one thread or process to another one. It allows multiple processes to share the same CPU. In context switching, the state of thread or process is stored so that the execution of the thread can be resumed later if required.

### 28. What is CyclicBarrier and CountDownLatch?

CyclicBarrier and CountDownLatch, both are required for managing multithreaded programming. But there is some difference between them as given below:

**CyclicBarrier:** It is a tool to synchronize threads processing using some algorithm. It enables a set of threads to wait for each other till they reach a common execution point or common barrier points, and then let them further continue execution. One can reuse the same CyclicBarrier even if the barrier is broken by setting it.

**CountDownLatch:** It is a tool that enables main threads to wait until mandatory operations are performed and completed by other threads. In simple words, it makes sure that a thread waits until the execution in another thread completes before it starts its execution. One cannot reuse the same CountDownLatch once the count reaches 0.



### 29. What do you mean by inter-thread communication?

Inter-thread communication, as the name suggests, is a process or mechanism using which multiple threads can communicate with each other. It is especially used to avoid thread polling in java and can be obtained using wait(), notify(), and notifyAll() methods.

### 30. What is Thread Scheduler and Time Slicing?

**Thread Scheduler**: It is a component of JVM that is used to decide which thread will execute next if multiple threads are waiting to get the chance of execution. By looking at the priority assigned to each thread that is READY, the thread scheduler selects the next run to execute. To schedule the threads, it mainly uses two mechanisms: Preemptive Scheduling and Time slicing scheduling.    
  
**Time Slicing**: It is especially used to divide CPU time and allocate them to active threads. In this, each thread will get a predefined slice of time to execute. When the time expires, a particular thread has to wait till other threads get their chances to use their time in a round-robin fashion. Every running thread will get executed for a fixed time period.

### 31. What is a shutdown hook?

A shutdown hook is simply a thread that is invoked implicitly before JVM shuts down. It is one of the most important features of JVM because it provides the capacity to do resource cleanup or save application state JVM shuts down.  By calling the halt(int) method of the Runtime class, the shutdown hook can be stopped. Using the following method, one can add a shutdown hook.

public void addShutdownHook(Thread hook){}

Runtime r=Runtime.getRuntime();

r.addShutdownHook(new MyThread());

### 32. What is busy spinning?

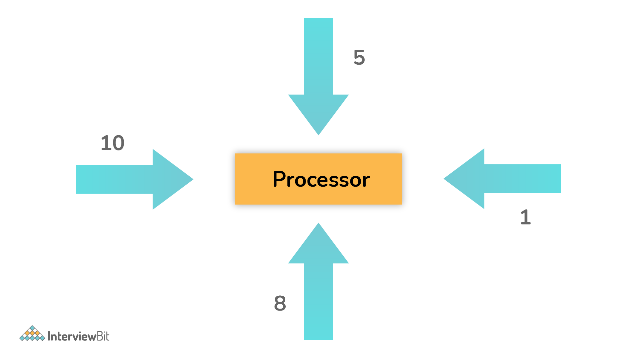
Busy Spinning, also known as Busy-waiting, is a technique in which one thread waits for some condition to happen, without calling wait or sleep methods and releasing the CPU. In this condition, one can pause a thread by making it run an empty loop for a certain time period, and it does not even give CPY control. Therefore, it is used to preserve CPU caches and avoid the cost of rebuilding cache.

### 33. What is ConcurrentHashMap and Hashtable? In java, why is ConcurrentHashMap considered faster than Hashtable?

**ConcurrentHashMap**: It was introduced in Java 1.5 to store data using multiple buckets. As the name suggests, it allows concurrent read and writes operations to the map. It only locks a certain portion of the map while doing iteration to provide thread safety so that other readers can still have access to the map without waiting for iteration to complete.    
  
**Hashtable**: It is a thread-safe legacy class that was introduced in old versions of java to store key or value pairs using a hash table.  It does not provide any lock-free read, unlike ConcurrentHashMap. It just locks the entire map while doing iteration.   
  
ConcurrentHashMap and Hashtable, both are thread-safe but ConcurrentHashMap generally avoids read locks and improves performance, unlike Hashtable. ConcurrentHashMap also provides lock-free reads, unlike Hashtable. Therefore, ConcurrentHashMap is considered faster than Hashtable especially when the number of readers is more as compared to the number of writers.

### 34. Explain thread priority.

Thread priority simply means that threads with the highest priority will get a chance for execution prior to low-priority threads. One can specify the priority but it's not necessary that the highest priority thread will get executed before the lower-priority thread. Thread scheduler assigns processor to thread on the basis of thread priority. The range of priority changes between 1-10 from lowest priority to highest priority.



### 35. What do you mean by the ThreadLocal variable in Java?

ThreadLocal variables are special kinds of variables created and provided by the Java ThreadLocal class. These variables are only allowed to be read and written by the same thread. Two threads cannot be able to see each other’s ThreadLocal variable, so even if they will execute the same code, then there won't be any race condition and the code will be thread-safe.    
  
**Example**:

public class ThreadLocalExp

{

public static class MyRunnable implements Runnable

{

private ThreadLocal<Integer> threadLocal =

new ThreadLocal<Integer>();

@Override

public void run() {

threadLocal.set( (int) (Math.random() \* 50D) );

try

{

Thread.sleep(1000);

} catch (InterruptedException e) {

}

System.out.println(threadLocal.get());

}

}

public static void main(String[] args)

{

MyRunnable runnableInstance = new MyRunnable();

Thread t1 = new Thread(runnableInstance);

Thread t2 = new Thread(runnableInstance);

// this will call run() method

t1.start();

t2.start();

}

}

**Output**:

10

33

10 33

### 36. What is semaphore?

Semaphore is regarded as a thread synchronization construct that is usually required to control and manage the access to the shared resource using counters. It simply sets the limit of the thread. The semaphore class is defined within the package java.util.concurrent and can be used to send signals between threads to avoid missed signals or to guard critical sections. It can also be used to implement resource pools or bounded collection.

### 37. Explain Thread Group. Why should we not use it?

ThreadGroup is a class that is used to create multiple groups of threads in a single object. This group of threads is present in the form of three structures in which every thread group has a parent except the initial thread. Thread groups can contain other thread groups also. A thread is only allowed to have access to information about its own thread group, not other thread groups.   
  
Previously in the old version of Java, the only functionality that did not work without a thread group was uncaughtException( Thread t, Throwable e). But now in Java 5 versions, there is Thread.setUncaughtExceptionHandler(UncaughtExceptionHandler). So now even that works without thread groups and therefore, there is no need to use thread groups.

t1.setUncaughtExceptionHandler(new UncaughtExceptionHandler()

{

@Override

public void uncaughtException(Thread t, Throwable e)

{

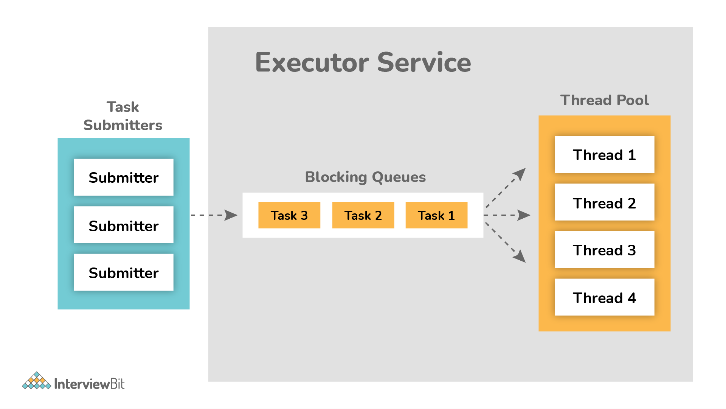
System.out.println("exception occured:"+e.getMessage());

}

};

### 38. What is the ExecutorService interface?

ExecutorService interface is basically a sub-interface of Executor interface with some additional methods or features that help in managing and controlling the execution of threads. It enables us to execute tasks asynchronously on threads.



**Example:**

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.TimeUnit;

public class TestThread {

public static void main(final String[] arguments) throws InterruptedException {

ExecutorService e = Executors.newSingleThreadExecutor();

try {

e.submit(new Thread());

System.out.println("Shutdown executor");

e.shutdown();

e.awaitTermination(5, TimeUnit.SECONDS);

} catch (InterruptedException ex) {

System.err.println("tasks interrupted");

} finally {

if (!e.isTerminated()) {

System.err.println("cancel non-finished tasks");

}

e.shutdownNow();

System.out.println("shutdown finished");

}

}

static class Task implements Runnable {

public void run() {

try {

Long duration = (long) (Math.random() \* 20);

System.out.println("Running Task!");

TimeUnit.SECONDS.sleep(duration);

} catch (InterruptedException ex) {

ex.printStackTrace();

}

}

}

}

**Output:**

Shutdown executor

shutdown finished

### 39. What will happen if we don’t override the thread class run() method?

Nothing will happen as such if we don’t override the run() method. The compiler will not show any error. It will execute the run() method of thread class and we will just don’t get any output because the run() method is with an empty implementation.   
  
**Example:**

class MyThread extends Thread {

//don't override run() method

}

public class DontOverrideRun {

public static void main(String[] args) {

System.out.println("Started Main.");

MyThread thread1=new MyThread();

thread1.start();

System.out.println("Ended Main.");

}

}

**Output:**

Started Main.

Ended Main.

### 40. What is the lock interface? Why is it better to use a lock interface rather than a synchronized block.?

Lock interface was introduced in Java 1.5 and is generally used as a synchronization mechanism to provide important operations for blocking.    
  
Advantages of using Lock interface over Synchronization block:

* Methods of Lock interface i.e., Lock() and Unlock() can be called in different methods. It is the main advantage of a lock interface over a synchronized block because the synchronized block is fully contained in a single method.
* Lock interface is more flexible and makes sure that the longest waiting thread gets a fair chance for execution, unlike the synchronization block.

### 41. Is it possible to call the run() method directly to start a new thread?

No, it's not possible at all. You need to call the start method to create a new thread otherwise run method won't create a new thread. Instead, it will execute in the current thread.

### 42. Is it possible that each thread can have its stack in multithreaded programming?

Of course, it is possible. In multithreaded programming, each thread maintains its own separate stack area in memory because of which every thread is independent of each other rather than dependent.

### When should one use Maven?

The Maven Build Tool can be used in the following conditions:

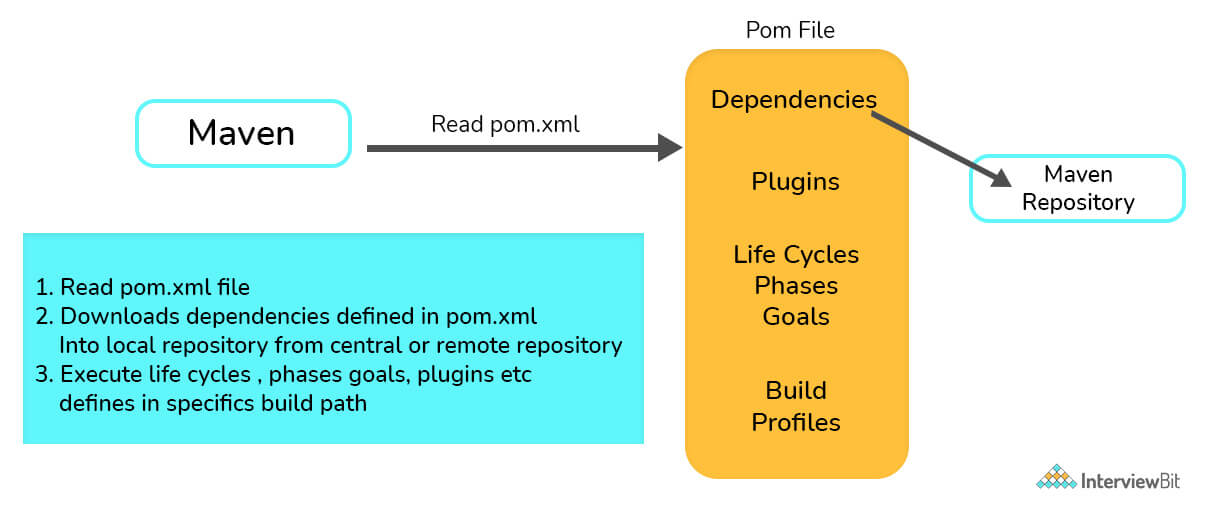
* When the project has a large number of dependencies. Then, using Maven, you can easily manage those dependencies.
* When the version of a dependency changes frequently. To update dependencies, simply update the version ID in the pom file.
* Maven makes it simple to handle continuous builds, integration, and testing.
* When you need a quick way to generate documentation from source code, this is the tool you use. It helps in compiling source code, and then packaging it into JAR or ZIP files.

### 2. Discuss the core concepts of Maven.

The core concepts of Maven are:

* **POM Files:** Project Object Model (POM) files are XML files that include information about the project and configuration information used by Maven to construct the project, such as dependencies, source directory, plugin, goals, and so on. When you want to run a maven command, you provide it with a POM file to run. To complete its configuration and functions, Maven reads the pom.xml file.
* **Dependencies and Repositories:** Repositories are folders containing bundled JAR files, and dependencies are external Java libraries necessary for Project. The local repository is simply a folder on your computer's hard drive. Maven retrieves dependencies from a central Maven repository and places them in your local repository if they aren't found in the local Maven repository.
* **Build Life Cycles, Phases, and Goals:** A build life cycle is made up of a series of build phases, each of which contains a set of goals. A build lifecycle, phase, or goal is referred to as a Maven command. When a lifecycle is asked to be run using the maven command, all of the build steps in that life cycle are likewise run. When a build phase is requested to be executed, it is followed by all build phases in the given sequence.
* **Build Profiles:** Build Profiles are a set of configuration parameters that allow you to build your project using a variety of setups. For example, you might need to develop and test your project on your local computer. You can add different build profiles to your POM files using its profile elements to enable different builds, which can be triggered in a variety of ways.
* **Build Plugins:** Build Plugins are used to accomplish a certain task. A plugin can be added to the POM file. Maven comes with various pre-installed plugins, but you can also write your own in Java.

### 3. How does Maven work?



Maven works in three steps:

* Reading the pom.xml file is the first step.
* The dependencies mentioned in pom.xml are then downloaded from the central repository into the local repository.
* Finally, it builds and generates a report based on the requirements, as well as handles life cycles, phases, goals, plugins, and other tasks.

You can download a PDF version of Maven Interview Questions.

[Download PDF](javascript:void(0))

### 4. List a few differences between Maven and ANT.

| **Ant** | **Maven** |
| --- | --- |
| Because Ant lacks formal conventions, we must include project structure information in the build.xml file. | Maven has a convention for storing source code, compiled code, and so forth. As a result, we don't need to provide project structure information in the pom.xml file. |
| Ant is procedural, so you'll need to write code to tell it what to do and when to do it. You must maintain order. | Maven has a convention for storing source code, compiled code, and so forth. As a result, we don't need to provide project structure information in the pom.xml file. |
| Ant has no life cycle. | Maven has a life cycle. |
| Ant is a toolbox. | Maven is a framework. |
| Ant is primarily a build tool. | Maven is primarily a project management tool. |
| The ant scripts can not be reused. | The maven plugins can be reused. |

### 5. What elements are used for creating a pom.xml file?

The following elements are necessary for creating a pom.xml file:

* **project**- The root element of the pom.xml file is the project.
* **modelVersion**- It identifies which version of the POM model you're working with. For Maven 2 and Maven 3, use version 4.0.0.
* **groupId**- groupId is the project group's identifier. It is unique, and you will most likely use a group ID that is similar to the project's root Java package name.
* **artifactId**- It is used for naming the project you're working on.
* **version**- The version number of the project is contained in the version element. If your project has been released in multiple versions, it is helpful to list the versions.

Other Pom.xml File Elements

* **dependencies**- This element is used to establish a project's dependency list.
* **dependency**- dependency is used inside the dependencies tag to define a dependency. The groupId, artifactId, and version of each dependency are listed.
* **name**- This element is used to give our Maven project a name.
* **scope**- This element is used to specify the scope of this maven project, which can include compile, runtime, test, among other things.
* **packaging**- The packaging element is used to package our project into a JAR, WAR, and other output formats.

### 6. What are the different types of Maven repositories? Discuss.

The three types of repositories of Maven are:

* Local repository
* Central repository
* Remote repository

Maven scans these repositories for dependencies. Maven looks in the Local repository first, then the Central repository, and finally the Remote repository if the Remote repository is defined in the POM.

* **Local Repository**: Local repository is a directory on the developer's device. The local repository contains all of Maven's dependencies. Even though several projects rely on dependencies, Maven only needs to download them once.
* **Central Repository**: The Maven community has built the central Maven repository. Maven searches this central repository for any dependencies that aren't available in your local repository. The dependencies are subsequently downloaded into your local repository by Maven.
* **Remote Repository**: Maven may download dependencies from a remote repository hosted on a web server. It is frequently used to host internal organization projects. The dependencies are subsequently downloaded into your local repository by Maven.

### 7. What command should one use to install JAR files in the Local Repository?

* JAR files are installed in the local repository using mvn install.
* The following plugin is used to manually install the JAR into the local Maven repository: install-file-Dfile = <file path>

### 8. In Maven, what do you mean by Clean, Default, and Site?

The three built-in build life cycles are:

* **Clean**: The clean lifecycle is in charge of project cleaning.
* **Default**: The project deployment is handled by the default lifecycle.
* **Site**: The creation of the project's site documentation is referred to as the site lifecycle.

### 9. What are the different phases of the default life cycle?

The different phases of the default lifecycle are:

* **Validate**: Make sure the project is correct and that you have all of the necessary information.
* **Test**: Test the compiled source code using an appropriate unit testing framework. These tests should not demand that the code be packed or deployed; instead, take the compiled code and package it in a manner that can be distributed, such as a JAR.
* **Compile:** Compile the project's source code.
* **Verify**: Perform any necessary checks on integration test findings to ensure that quality criteria are met.
* **Install:** Adds the package to the local repository, allowing it to be used as a dependency in other projects.
* **Deploy:** Copies the entire package to the remote repository for sharing with other developers and organizations, and is done in the build environment.

### 10. What are Maven plugins used for? What are the types of Maven plugins?

Maven Plugins are used for:

* Creating JAR files.
* Creating WAR files.
* Compiling the source code files.
* Unit testing of the code.
* Creating the project documentation.
* Creating project reports.

Maven plugins are divided into two categories:

* **Build plugins:**  These plugins are used throughout the build process and are configured in the pom.xml file's <build/> element.
* **Reporting plugins:** These plugins are configured in the pom.xml's <reporting/> element and run during stage generation.

### 11. “Maven uses convention over configuration” - Why is it so?

* Because developers simply need to establish a Maven project while employing convention,  Maven uses convention over configuration. The rest of the structure is generated automatically. In the case of configuration, build processes are to be created manually.
* For setting up a project, creating artifacts, releasing code, and running unit tests, Maven has a number of conventions.

### 12. What is Maven's inheritance order?

In Maven, the order of inheritance is:

* Settings
* CLI parameters
* Parent POM
* Project POM

### 13. In Maven, what is a snapshot?

A snapshot is a specific version of a project that shows the most recent development copy of the project being worked on. Maven always checks out a SNAPSHOT of the project in the remote repository for each build.

As a result, anytime Maven discovers a newer SNAPSHOT of the project, it downloads and replaces the project's older .jar file in the local repository.

### 14. What are the locations where Maven dependencies are stored?

Maven saves all of the JARs, dependency files, and other things it downloads in the Maven local repository. All of the artifacts are kept locally in the Maven local repository, which is a folder on the local machine.

### 15. What are the different types of Maven build profiles? In what ways can build profiles of maven be activated?

The different types of Maven build profiles are:

* **Per-User**: This is defined in the Maven settings.xml file.
* **Per Project**: This is defined in the project’s pom.xml.
* **Global**: This is defined in the global Maven settings.xml file.

Maven build profiles can be activated or triggered in the following ways:

* Using explicit commands
* Maven settings
* On the basis of environment variables
* Configuration of the operating system
* Present/missing files

### 16. How would you refer to a property declared in your pom.xml file?

In order to refer to a property declared in your pom.xml, the property name makes use of the names of the XML components that designate the value, with "pom" being accepted as a synonym for the project (root) element.

So ${pom.name} is the project's name, ${pom.version} is the project's version, ${pom.build.finalName} is the final name of the file generated when the built project is packaged, and so on.

### 17. How to generate javadocs in Maven?

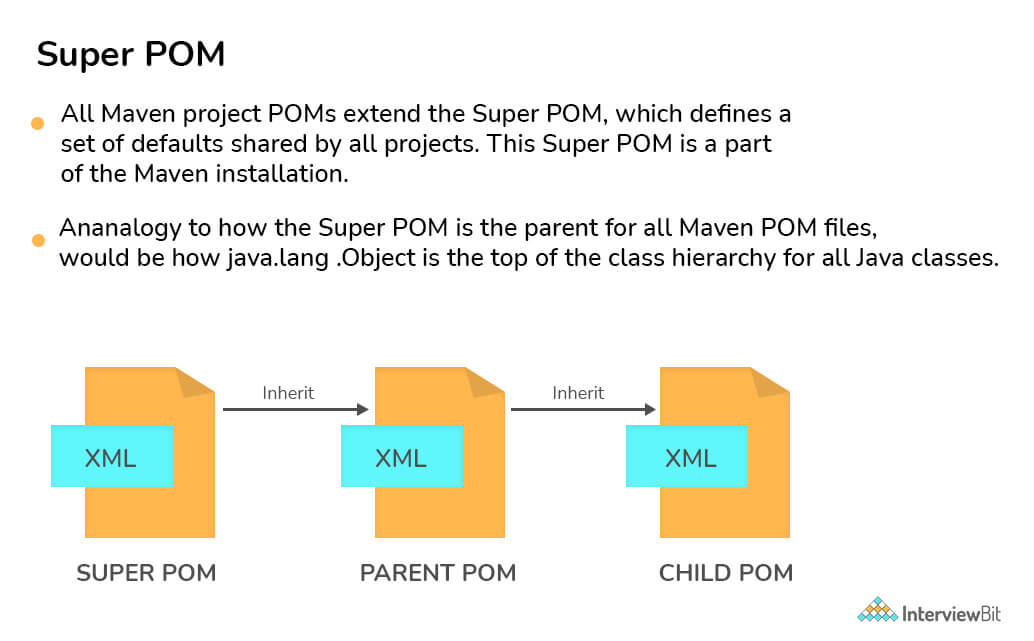
The maven-javadoc plugin is used by Maven to generate a project's javadocs. To create javadocs, this plugin internally uses the JDK\bin\javadoce.exe command. The javadocs for the project are generated when the project is deployed with the mvn install command.

### 18. What exactly is MOJO?

Every Maven plain Old Java Object (MOJO) is an executable goal, and a plugin pertains to the distribution of these MOJOs. MOJO allows Maven to add functionalities that it doesn't already have. In Maven, a MOJO is a single unit of the task.

### 19. What do you understand about the term ‘Super POM’?

Any POM file has the ability to point to its parent POM. There is a system-wide POM file that is automatically considered as the parent POM file if the parent POM element is absent. The super POM is the name given to this POM file. Finally, the super POM is used to extend all of the application POM files. The apex of the POM hierarchy is the super POM file. The super POM file contains all of the default configurations. All of the configurations defined in the super POM file will be inherited by even the simplest version of a POM file. You can alter any option you want by redefining the same section in your application POM file.



### 20. What is a 'Dependency Scope'? What are the different types of Dependency Scopes?

The dependency scope pertains to all dependencies related to the present stage of the build.

The following are the several sorts of dependence scopes:

* **Compile**- It's the default scope, and it shows which dependencies are available in the project's classpath.
* **Provided**- It denotes that the dependency is delivered at runtime by the JDK, web server, or container.
* **Runtime**- This indicates that the dependency is not required during compilation but is necessary during execution.
* **Test**- It claims that dependencies are only available during the test compilation and execution phases.
* **System**- It implies that you must specify the system path.
* **Import**- This means that the dependencies in that POM's section should be used instead of the identified or specified POM.

## Maven Interview Questions for Experienced

### 21. What do you mean by a Maven Archetype? How will you create a new project based on an Archetype?

Maven Archetype is a Maven plugin that makes it possible to create a project structure based on a template. These archetypes are essentially project templates that Maven generates when you create a new project. Archetype is a Maven project templating toolkit, in a nutshell.

After getting to the directory where the project is located, type the command: – **mvn archetype: generate** in the command prompt. This aids in creating a new project based on an archetype.

There are four steps for creating a project from an archetype:

* prepare a repository reference
* the choice of an archetype,
* that archetype's configuration,
* the efficient creation of the project using the data gathered

In most cases, an archetype is procured from a remote repository. You're ready to go if that repository can be reached using your Maven configuration. You must add the repository to your settings.xml if the repository is not managed and you wish to refer to it directly.

### 22. What command is used to create a new project from a hard drive?

The **-mvn archetype: create** is used to start a new project.  
After reading the source and resource files, as well as the values of its parameters and other properties, the archetype is constructed.

### 23. What are the phases of the clean lifecycle?

The Maven clean lifecycle takes care of eliminating all temporary files from the output directory, including generated source files, compiled classes, and previous JAR files, among other things.

* pre-clean- performs tasks that are necessary prior to actual project cleaning.
* clean- delete all files created by the previous build.
* post-clean- performs tasks that are necessary to finalize project cleaning.

### 24. What are the phases of the site lifecycle?

Everything related to generating documentation for your project is handled by the Maven site lifecycle.

* **pre-site**- performs tasks that are necessary prior to actual project site generation.
* **site**- develop the project’s site generation.
* **post-site**- performs tasks that are necessary to finalize project site generation, also prepares for site deployment.
* **site-deploy**- deploy the developed site documentation to the web server of your choice.

### 25. Explain the three commonly used plugins: clean, surefire, antrun.

Maven **clean** is a plugin that, as the name implies, attempts to clean the files and directories generated by Maven during the build process. The target folder, which contains all of the class files, documentation, and JAR files, is removed by the plugin.

The **Surefire** Plugin is used to run an application's unit tests during the test phase of the build lifecycle. It can generate reports in one of two file formats: plain text files or XML files.

The **Antrun** Plugin allows you to perform Ant tasks directly from within Maven. Your Ant scripts can even be embedded in the POM!

### 26. What is the settings.xml file in Maven?

A Maven installation is configured using the settings.xml file. It's comparable to a pom.xml file, but it's either global or user-specific. The Maven settings.xml file provides elements that define the values required to configure Maven's execution in several ways. These values include the location of the local repository, authentication information, and alternate remote repository servers among others.

Let's look at the elements in the settings.xml file that we can change. The settings.xml file's main element, settings, can include up to nine predefined child elements:

<settings xmlns = "http://maven.apache.org/SETTINGS/1.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 https://maven.apache.org/xsd/settings-1.0.0.xsd">

<localRepository/>

<interactiveMode/>

<offline/>

<pluginGroups/>

<servers/>

<mirrors/>

<proxies/>

<profiles/>

<activeProfiles/>

</settings>

The following configurations are included:

* Proxy configuration
* Local repository configuration
* Remote repository configuration
* Central repository configuration

### 27. What is dependency mediation and dependency management?

When multiple versions of an artifact are encountered, Maven determines which version of the dependency should be used. The earliest declared dependence will be used if two dependency versions are at the same depth in the dependency tree. This is referred to as "dependency mediation."

Dependency management allows project authors to declare the versions of artifacts that are to be utilized when they are discovered in transitive dependencies or dependencies that have no version specified.

### 28. What do you mean by the term “system dependency”?

The term "system dependency" refers to the scope system's dependency. These dependencies are typically used to let Maven know the dependencies the JDK or VM provides. System dependencies are typically used to resolve dependencies on JDK-provided artifacts. Some common examples are the Java Authentication and Authorization Service (JAAS) or the JDBC standard extensions.

### 29. What is the use of an Optional Dependency?

When splitting a project into submodules isn't practicable (for some reason), optional dependencies are employed. The concept is that some of the dependencies are just required for particular project features and will not be required if those features are not used. Such a feature should ideally be divided into a sub-module that is dependent on the project's main functionality. Only non-optional dependencies would be included in this new subproject, as you'd need them all if you wanted to use the subproject's features.

If a user wants to use functionality associated with an optional dependency, they must redeclare it in their own project. Optional dependencies save storage and memory. They prevent troublesome jars from being packed into a WAR, EAR, fat jar, or other formats if they violate a license agreement or cause classpath difficulties.

### 30. What do you understand about ‘Transitive Dependency’ in Maven? What is dependency exclusion?

By incorporating transitive dependencies automatically, Maven eliminates the need to discover and define libraries that the dependencies require. According to transitive dependency, if X is dependent on Y and Y is dependent on Z, then X is dependent on both Y and Z.

The "exclusion" element can be used to exclude any transitive dependency. If X is reliant on Y and Y is reliant on Z, then X can declare Z as excluded.

### 31. What are the elements that must be defined for each external dependency?

The Maven software relies heavily on external dependencies. It is an intrinsic component of the system without which it is impossible to find dependencies in a system. We'll need the following information to specify the external dependency:

* It necessitates a group ID that is identical to the library name.
* It necessitates an artifact ID that is identical to the library name.
* The system's dependency scope must be mentioned.
* The system path that corresponds to the project position must be mentioned.

### 32. What are user-defined properties?

You have the opportunity to define your own arbitrary properties in addition to the implicit properties. A POM or a Profile can be used to define properties. The properties defined in a POM or a Maven Profile can be referenced in Maven just like any other property. User-defined properties can be used to filter resources via the Maven Resource plugin, or they can be referenced in a POM. In a Maven POM, here's an example of defining some arbitrary properties.

<project>

...

<properties>

<arbitrary.property.x>Text</arbitrary.property.x>

<hibernate.version>3.2.1.ga</hibernate.version>

</properties>

...

<dependencies>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate</artifactId>

<version>${hibernate.version}</version>

</dependency>

</dependencies>

...

</project>

arbitrary.property.x and hibernate.version are two properties defined in the preceding example. In a dependency declaration, hibernate.version is mentioned. It's usual practice in Maven POMs and Profiles to use the period character as a separator in property names. The following example demonstrates how to define a property in a Maven POM profile.

<project>

...

<profiles>

<profile>

<id>random-profile</id>

<properties>

<arbitrary.property>Text</arbitrary.property>

</properties>

</profile>

</profiles>

...

</project>

### 33. Discuss the profile element in settings.xml file.

The settings.xml profile element is a trimmed version of the pom.xml profile element. It is made up of the elements: activation, repositories, pluginRepositories, and properties. These four components are the only ones included in the profile elements since they deal with the build system as a whole (which is what the settings.xml file is for), not individual project object model settings.

If a profile is activated from settings, its values will override any POM or profiles.xml profiles with the same ID.

* **Activation**: The strength of a profile, like that of the POM's profiles, comes from its capacity to modify specific values only under certain conditions, which are stated via an activation element.
* **Repositories**: Repositories are remote collections of projects that Maven utilizes to populate the build system's local repository.
* **pluginRepositories**: Maven plugins are a unique form of artifact in themselves. Plugin repositories may be segregated from other repositories as a result of this. Each of the pluginRepository components specifies a remote source where Maven can look for new plugins.
* **Properties**: Maven properties, like Ant properties, value placeholders. The notation ${X}, where X is the property, can be used to obtain their values anywhere within a POM.

### 34. What is maven-release plugin and how does it work?

The maven release plugin is used to automate the build and release process. When maven executes the maven-release-plugin, the following activities are performed:

* **mvn release:clean** - clears the workspace from the previous build and prepares it for a new one.
* **mvn release:rollback** - If the previous process failed, it rollbacks the workspace.
* **mvn release:prepare** - It performs the following tasks:
  + Checks the local workspace for any uncommitted files.
  + Checks for SNAPSHOT dependencies and verifies they aren't present.
  + Prepares the final version for release.
  + Updates the pom to SCM (SVN/Git/Mercurial/CVS).
  + Runs the test cases.
  + Executes the ultimate commit to the SCM.
  + Tags the script/code.
  + Increases the version number and includes the SNAPSHOT as part of the subsequent releases.
* **mvn release:perform** - fetches the code from the repository and executes the maven goal to develop and deploy the artifacts.

### 35. Why are exclusions made on a dependency-by-dependency basis instead of at the POM level?

This is primarily to ensure that the dependency graph is predictable, as well as to prevent inheritance effects from eliminating a dependent that should not be excluded. If you have to use the method of last resort and add an exclusion, make sure you know which of your dependencies is causing the undesirable transitive dependency.

The banned dependencies rule can be specified to fail the build if a troublesome dependency is identified, regardless of path. You'll need to add specific exclusions to each path the enforcer detects if the build fails.

### 36. Explain the default and the advanced configuration inheritance.

The default behavior includes merging the content of the configuration element according to the element name. If a certain element exists in the child POM, that value becomes the effective value. The parent value becomes the effective value if the child POM does not have an element but the parent does. It's important to note that this is solely an XML operation, with no code or plugin settings involved. Only the elements are involved, not their values.

Advanced configuration inheritance includes adding attributes to the children of the configuration element to regulate how child POMs inherit configuration from parent POMs. Combine.children and combine.self are the two attributes. These attributes can be used in a child POM to regulate how Maven integrates the parent's plugin configuration with the child's explicit configuration.

### 37. Explain Project Aggregation.

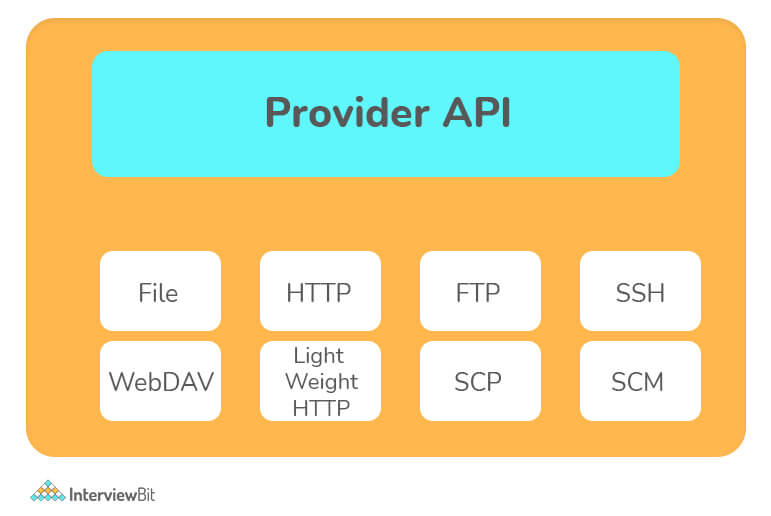
Project Aggregation specifies the modules from the parent POM instead of specifying the parent POM from the module. As a result, the parent project is aware of its modules, and if a Maven command is issued against the parent project, the Maven command is also applied to the parent's modules. For Project Aggregation, you must accomplish the following:

* Change the packaging of the parent POMs to "pom."
* Specify the modules' directories in the parent POM (children POMs).

### 38. What is the use of the Maven Wagon plugin?

The Maven Wagon Plugin, as its name suggests, allows you to access numerous Maven Wagon functionalities. To transfer resources to and from Maven repositories, Maven Wagon offers a layer of abstraction over the core transport protocols. Maven Wagon's unified API includes implementations for seven transports.

The following picture depicts the architecture of the Maven Wagon:



The plugin allows you to use the wagon to upload resources from your build to a remote site, get resources from a repository and list the contents of a repository. Finally, it may merge a Maven repository to another in a generic fashion by merging the upload and download capabilities.

### 39. How is Doxia used by Maven?

Doxia is a content creation framework that aims to give powerful approaches for creating static and dynamic content to its users: Doxia can be used to create static web pages in a web-based publication context, as well as in dynamic content creation systems such as blogs, wikis, and content management systems.

Maven makes substantial use of Doxia, which powers the project's complete documentation system. It enables Maven to take any Doxia-supported document and output it in any format.

For instance, 'mvn site' is the command used by Maven to produce javadocs for a specific project. Maven calls Doxia document generation and other report generating plugins when this command is run.

### 40. How will you run JUnit tests in parallel with a Maven build?

It is now possible to run tests in parallel without utilizing TestNG in junit 4.7. It's been feasible since 4.6, but 4.7 will include a number of improvements that will make it a realistic alternative. You may also use spring to execute parallel tests.

You can also use this maven plugin:

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>2.6.0</version>

<configuration>

<parallel>classes</parallel>

<threadCount>4</threadCount>

</configuration>

</plugin>

</plugins>

</build>

### 41. How can you skip running the tests for a particular project?

Set the skipTests attribute to true to skip the tests for a certain project.

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>2.13.0</version>

<configuration>

<skipTests>true</skipTests>

</configuration>

You may also skip the tests by using the following command from the command line:

mvn install -DskipTests

You can also use the maven.test.skip option to avoid compiling the tests if you have to. Surefire, Failsafe, and the Compiler Plugin all recognize maven.test.skip.

mvn install -Dmaven.test.skip=true

You must go through a properties section in the pom if you want to skip tests by default but have the possibility to re-enable tests from the command line:

<properties>

<skipTests>true</skipTests>

</properties>

And,

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>2.13.0</version>

<configuration>

<skipTests>${skipTests}</skipTests>

</configuration>

This will allow you to run tests with the default tests disabled and execute them with the following command:

mvn install -DskipTests=false

The "skip" parameter and other booleans on the plugin can be used in the same way.

### 42. What is the difference between the maven package and the maven install?

**package:** converts the compiled code into a distributable format, such as a JAR.

**install:** adds the package to the local repository, allowing it to be used as a dependency in other projects.

### ****What are the major features in different versions of Spring Framework?****

|  |  |  |
| --- | --- | --- |
| ****Features of Spring Framework**** | | |
| **Version** | **Logo** | **Feature** |
| **Spring 2.5** | spring 2.5 logo - Spring Interview Questions - Edureka! | This version was released in 2007. It was the first version which supported annotations. |
| **Spring 3.0** | spring 3.0 logo - Spring Interview Questions - Edureka! | This version was released in 2009. Itmade full-fledged use of improvements in Java5 and also provided support to JEE6. |
| **Spring 4.0** | Spring 4.0 logo - Spring Interview Questions - Edureka! | This version was released in 2013. This was the first version to provide full support to Java 8. |

### ****2. What is a Spring Framework?****

* Spring is a powerful open source, application framework created to reduce the complexity of enterprise application development.
* It is light-weighted and loosely coupled.
* It has layered architecture, which allows you to select the components to use, while also providing a cohesive framework for J2EE application development.
* Spring framework is also called the framework of frameworks as it provides support to various other frameworks such as Struts, Hibernate, Tapestry, EJB, JSF etc.

### ****3. List the advantages of Spring Framework.****

* Because of Spring Frameworks layered architecture, you can use what you need and leave which you don’t.
* Spring Framework enables POJO (Plain Old Java Object) Programming which in turn enables continuous integration and testability.
* JDBC is simplified due to Dependency Injection and Inversion of Control.
* It is open-source and has no vendor lock-in.

### ****4. What are the different features of Spring Framework?****

Following are some of the major features of Spring Framework :

* **Lightweight:** Spring is lightweight when it comes to size and transparency.
* **Inversion of control (IOC):** The objects give their dependencies instead of creating or looking for dependent objects. This is called Inversion Of Control.
* **Aspect oriented Programming (AOP):** Aspect oriented programming in Spring supports cohesive development by separating application business logic from system services.
* **Container:**Spring Framework creates and manages the life cycle and configuration of the application objects.
* **MVC Framework:** Spring Framework’s MVC web application framework is highly configurable. Other frameworks can also be used easily instead of Spring MVC Framework.
* **Transaction Management:** Generic abstraction layer for transaction management is provided by the Spring Framework. Spring’s transaction support can be also used in container less environments.
* **JDBC Exception Handling:** The JDBC abstraction layer of the Spring offers an exception hierarchy, which simplifies the error handling strategy.

### ****5. How many modules are there in Spring Framework and what are they?****

There are around 20 modules which are generalized into Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation and Test.

* **Spring Core Container –** This layer is basically the core of Spring Framework.It contains the following modules :

1. Spring Core
2. Spring Bean
3. SpEL (Spring Expression Language)
4. Spring Context

* **Data Access/Integration –** This layer provides support to interact with the database. It contains the following modules :

1. JDBC (Java DataBase Connectivity)
2. ORM (Object Relational Mapping)
3. OXM (Object XML Mappers)
4. JMS (Java Messaging Service)
5. Transaction

* **Web –**This layer provides support to create web application. It contains the following modules :

1. Web
2. Web – MVC
3. Web – Socket
4. Web – Portlet

* **Aspect Oriented Programming (AOP) –** In this layer you can use Advices, Pointcuts etc., to decouple the code.
* **Instrumentation –**This layer provides support to class instrumentation and classloader implementations.
* **Test –**This layer provides support to testing with JUnit and TestNG.

Few Miscellaneous modules are given below:

* **Messaging –**This module provides support for STOMP. It also supports an annotation programming model that is used for routing and processing STOMP messages from WebSocket clients.
* **Aspects –**This module provides support to integration with AspectJ.

### ****6. What is a Spring configuration file?****xml file - Spring Interview Questions - Edureka!

A Spring configuration file is an XML file. This file mainly contains the classes information. It describes how those classes are configured as well as introduced to each other. The XML configuration files, however, are verbose and more clean. If it’s not planned and written correctly, it becomes very difficult to manage in big projects.

### ****7. What are the different components of a Spring application?****

A Spring application, generally consists of following components:

* Interface: It defines the functions.
* Bean class: It contains properties, its setter and getter methods, functions etc.
* Spring Aspect Oriented Programming (AOP): Provides the functionality of cross-cutting concerns.
* Bean Configuration File: Contains the information of classes and how to configure them.
* User program: It uses the function.

### ****8. What are the various ways of using Spring Framework?****spring app - Spring Interview Questions - Edureka!

Spring Framework can be used in various ways. They are listed as follows:

1. As a Full-fledged Spring web application.
2. As a third-party web framework, using Spring Frameworks middle-tier.
3. For remote usage.
4. As Enterprise Java Bean which can wrap existing POJOs (Plain Old Java Objects).

The next section of Spring Interview Questions is on Dependency Injection and IoC container.

## **Dependency Injection/ IoC Container – Spring Interview Questions**

### ****9. What is Spring IOC Container?****

## ioc - Spring Interview Questions - Edureka!

At the core of the Spring Framework, lies the Spring container. The container creates the object, wires them together, configures them and manages their complete life cycle. The Spring container makes use of Dependency Injection to manage the components that make up an application. The container receives instructions for which objects to instantiate, configure, and assemble by reading the configuration metadata provided. This metadata can be provided either by XML, Java annotations or Java code.

### ****10. What do you mean by Dependency Injection?****

In Dependency Injection, you do not have to create your objects but have to describe how they should be created. You don’t connect your components and services together in the code directly, but describe which services are needed by which components in the configuration file. The IoC container will wire them up together.

### ****11. In how many ways can Dependency Injection be done?****

In general, dependency injection can be done in three ways, namely :

* Constructor Injection
* Setter Injection
* Interface Injection

In Spring Framework, only constructor and setter injections are used.

### ****12. Differentiate between constructor injection and setter injection.****

#### Constructor Injection vs Setter Injection

|  |  |
| --- | --- |
| **Constructor Injection** | **Setter Injection** |
| There is no partial injection. | There can be partial injection. |
| It doesn’t override the setter property. | It overrides the constructor property. |
| It will create a new instance if any modification is done. | It will not create new instance if any modification is done. |
| It works better for many properties. | It works better for few properties. |

### ****13. How many types of IOC containers are there in spring?****

1. **BeanFactory**: BeanFactory is like a factory class that contains a collection of beans. It instantiates the bean whenever asked for by clients.
2. **ApplicationContext**: The ApplicationContext interface is built on top of the BeanFactory interface. It provides some extra functionality on top BeanFactory.

### ****14. Differentiate between BeanFactory and ApplicationContext.****

#### BeanFactory vs ApplicationContext

|  |  |
| --- | --- |
| **BeanFactory** | **ApplicationContext** |
| It is an interface defined in org.springframework.beans.factory.**BeanFactory** | It is an interface defined in org.springframework.context.**ApplicationContext** |
| It uses Lazy initialization | It uses Eager/ Aggressive initialization |
| It explicitly provides a resource object using the syntax | It creates and manages resource objects on its own |
| It doesn’t supports internationalization | It supports internationalization |
| It doesn’t supports annotation based dependency | It supports annotation based dependency |

### ****15.  List some of the benefits of IoC.****

Some of the benefits of IoC are:

* It will minimize the amount of code in your application.
* It will make your application easy to test because it doesn’t require any singletons or JNDI lookup mechanisms in your unit test cases.
* It promotes loose coupling with minimal effort and least intrusive mechanism.
* It supports eager instantiation and lazy loading of the services.

Let’s move on to the next section of Spring Interview Questions, that is Spring Beans Interview Questions.

## **Spring Beans – Spring Interview Questions**

### ****16. Explain Spring Beans?****

* They are the objects that form the backbone of the user’s application.
* Beans are managed by the Spring IoC container.
* They are instantiated, configured, wired and managed by a Spring IoC container
* Beans are created with the configuration metadata that the users supply to the container.

### ****17. How configuration metadata is provided to the Spring container?****

Configuration metadata can be provided to Spring container in following ways:

* **XML-Based configuration:**In Spring Framework, the dependencies and the services needed by beans are specified in configuration files which are in XML format. These configuration files usually contain a lot of bean definitions and application specific configuration options. They generally start with a bean tag. For example:

|  |  |
| --- | --- |
| 1  2  3 | <bean id="studentbean" class="org.edureka.firstSpring.StudentBean">   <property name="name" value="Edureka"></property>  </bean> |

* **Annotation-Based configuration**: Instead of using XML to describe a bean wiring, you can configure the bean into the component class itself by using annotations on the relevant class, method, or field declaration. By default, annotation wiring is not turned on in the Spring container. So, you need to enable it in your Spring configuration file before using it. For example:

|  |  |
| --- | --- |
| 1  2  3  4 | <beans>  <context:annotation-config/>  <!-- bean definitions go here -->  </beans> |

* **Java-based configuration:**The key features in Spring Framework’s new Java-configuration support are @Configuration annotated classes and @Bean annotated methods.

1. @Bean annotation plays the same role as the <bean/> element.

2.@Configuration classes allows to define inter-bean dependencies by simply calling other @Bean methods in the same class.

For example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | @Configuration  public class StudentConfig  {  @Bean  public StudentBean myStudent()  { return new StudentBean(); }  } |

### ****18. How many bean scopes are supported by Spring?****

The Spring Framework supports five scopes. They are:

* **Singleton:**This provides scope for the bean definition to single instance per Spring IoC container.
* **Prototype:**This provides scope for a single bean definition to have any number of object instances.
* **Request:**This provides scope for a bean definition to an HTTP-request.
* **Session:**This provides scope for a bean definition to an HTTP-session.
* **Global-session:**This provides scope for a bean definition to an Global HTTP-session.

The last three are available only if the users use a web-aware ApplicationContext.

### ****19. What is the Bean life cycle in Spring Bean Factory Container?****

Bean life cycle in Spring Bean Factory Container is as follows:

1. The Spring container instantiates the bean from the bean’s definition in the XML file.
2. Spring populates all of the properties using the dependency injection, as specified in the bean definition.
3. The factory calls setBeanName() by passing the bean’s ID, if the bean implements the BeanNameAware interface.
4. The factory calls setBeanFactory() by passing an instance of itself, if the bean implements the BeanFactoryAware interface.
5. preProcessBeforeInitialization() methods are called if there are any BeanPostProcessors associated with the bean.
6. If an init-method is specified for the bean, then it will be called.
7. Finally, postProcessAfterInitialization() methods will be called if there are any BeanPostProcessors associated with the bean.

To understand it in better way check the below diagram:



### ****20. Explain inner beans in Spring.****

A bean can be declared as an inner bean only when it is used as a property of another bean. For defining a bean, the Spring’s XML based configuration metadata provides the use of <bean> element inside the <property> or <constructor-arg>. Inner beans are always anonymous and they are always scoped as prototypes. For example, let’s say we have one Student class having reference of Person class. Here we will be creating only one instance of Person class and use it inside Student.

Here’s a Student class followed by bean configuration file:

Student.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | public class Student  {  private Person person;  //Setters and Getters  }  public class Person  {  private String name;  private String address;  //Setters and Getters  } |

studentbean.xml

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <bean id=“StudentBean" class="com.edureka.Student">  <property name="person">  <!--This is inner bean -->  <bean class="com.edureka.Person">  <property name="name" value=“Scott"></property>  <property name="address" value=“Bangalore"></property>  </bean>  </property>  </bean> |

### ****21. Define Bean Wiring.****

When beans are combined together within the Spring container, it’s called wiring or bean wiring. The Spring container needs to know what beans are needed and how the container should use dependency injection to tie the beans together, while wiring beans.



### ****22. What do you understand by auto wiring and name the different modes of it?****

The Spring container is able to autowire relationships between the collaborating beans. That is, it is possible to let Spring resolve collaborators for your bean automatically by inspecting the contents of the BeanFactory.  
Different modes of bean auto-wiring are:

1. **no:** This is default setting which means no autowiring. Explicit bean reference should be used for wiring.
2. **byName:** It injects the object dependency according to name of the bean. It matches and wires its properties with the beans defined by the same names in the XML file.
3. **byType:** It injects the object dependency according to type. It matches and wires a property if its type matches with exactly one of the beans name in XML file.
4. **constructor:** It injects the dependency by calling the constructor of the class. It has a large number of parameters.
5. **autodetect:** First the container tries to wire using autowire by *constructor*, if it can’t then it tries to autowire by *byType*.

### ****23. What are the limitations with auto wiring?****

Following are some of the limitations you might face with auto wiring:

* **Overriding possibility:**You can always specify dependencies using <constructor-arg> and <property> settings which will override autowiring.
* **Primitive data type:** Simple properties such as primitives, Strings and Classes can’t be autowired.
* **Confusing nature:** Always prefer using explicit wiring because autowiring is less precise.

In the next section, we will discuss on Spring Annotations Interview Questions.

## **Spring Annotations – Spring Interview Questions**

### ****24. What do you mean by  Annotation-based container configuration?****

Instead of using XML to describe a bean wiring, the developer moves the configuration into the component class itself by using annotations on the relevant class, method, or field declaration. It acts as an alternative to XML setups. For example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | @Configuration  public class AnnotationConfig  {  @Bean  public MyDemo myDemo()   { return new MyDemoImpll(); }  } |

### ****25. How annotation wiring can be turned on in Spring?****

By default, Annotation wiring is not turned on in the Spring container. Thus, to use annotation based wiring we must enable it in our Spring configuration file by configuring **<context:annotation-config/>** element. For example:

|  |  |
| --- | --- |
| 1  2  3  4 | <beans xmlns="<a href="http://www.springframework.org/schema/beans">http://www.springframework.org/schema/beans</a>" xmlns:xsi="<a href="http://www.w3.org/2001/XMLSchema-instance">http://www.w3.org/2001/XMLSchema-instance</a>" xmlns:context="<a href="http://www.springframework.org/schema/context">http://www.springframework.org/schema/context</a>">  <context:annotation-config/>  <beans ………… />  </beans> |

### ****26. What’s the difference between @Component, @Controller, @Repository & @Service annotations in Spring?****annotations - Spring Framework Tutorial - Edureka!

**@Component:** This marks a java class as a bean. It is a generic stereotype for any Spring-managed component. The component-scanning mechanism of spring now can pick it up and pull it into the application context.

**@Controller:** This marks a class as a Spring Web MVC controller. Beans marked with it are automatically imported into the Dependency Injection container.

**@Service:** This annotation is a specialization of the component annotation. It doesn’t provide any additional behavior over the @Component annotation. You can use @Service over @Component in service-layer classes as it specifies intent in a better way.

**@Repository:** This annotation is a specialization of the @Component annotation with similar use and functionality. It provides additional benefits specifically for DAOs. It imports the DAOs into the DI container andmakes the unchecked exceptions eligible for translation into Spring DataAccessException.

### ****27. What do you understand by @Required annotation?****

@Required is applied to bean property setter methods. This annotation simply indicates that the affected bean property must be populated at the configuration time with the help of an explicit property value in a bean definition or with autowiring. If the affected bean property has not been populated, the container will throw BeanInitializationException.

For example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | public class Employee  {  private String name;  @Required  public void setName(String name)  {this.name=name; }  public string getName()  { return name; }  } |

### ****28. What do you understand by @Autowired annotation?****

The **@Autowired** annotation provides more accurate control over where and how autowiring should be done. This annotation is used to autowire bean on the setter methods, constructor, a property or methods with arbitrary names or multiple arguments. By default, it is a type driven injection.

For Example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | public class Employee  {  private String name;  @Autowired  public void setName(String name)  {this.name=name; }  public string getName()  { return name; }  } |

### ****29. What do you understand by @Qualifier annotation?****

When you create more than one bean of the same type and want to wire only one of them with a property  you can use the **@Qualifier** annotation along with **@Autowired** to remove the ambiguity by specifying which exact bean should be wired.

For example, here we have two classes, Employee and EmpAccount respectively. In EmpAccount, using @Qualifier its specified that bean with id emp1 must be wired.

Employee.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | public class Employee  {  private String name;  @Autowired  public void setName(String name)  { this.name=name; }  public string getName()  { return name; }  } |

EmpAccount.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | public class EmpAccount  {  private Employee emp;  @Autowired  @Qualifier(emp1)  public void showName()  {  System.out.println(“Employee name : ”+emp.getName);  }  } |

### ****30.  What do you understand by @RequestMapping annotation?****

@RequestMapping annotation is used for mapping a particular HTTP request method to a specific class/ method in controller that will be handling the respective request. This annotation can be applied at both levels:

* **Class level** : Maps the URL of the request
* **Method level**: Maps the URL as well as HTTP request method

Next section of Spring Interview Questions is on Data Access.

## **Data Access – Spring Interview Questions**

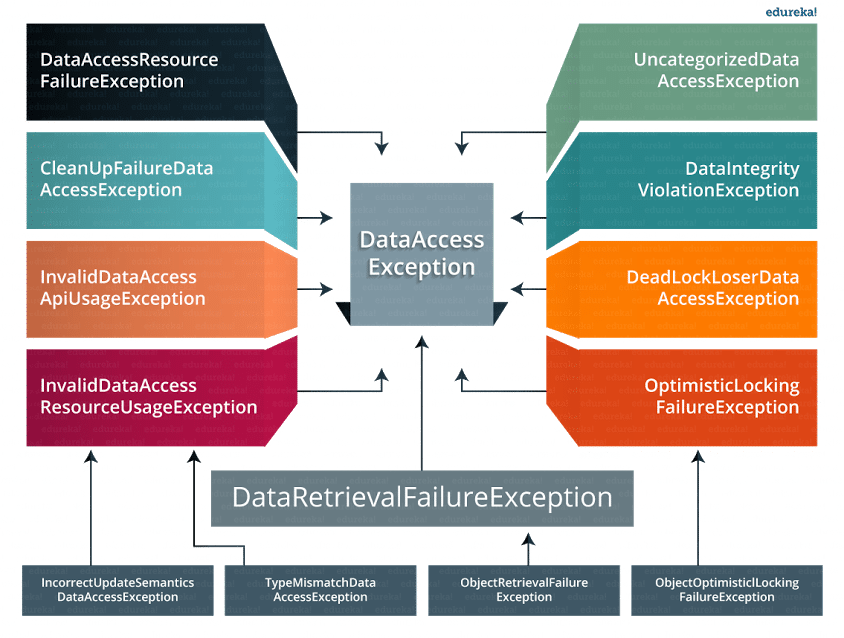
### ****31. Describe Spring DAO support?****

The Data Access Object (DAO) support in Spring makes it easy to work with data access technologies like JDBC, Hibernate or JDO in a consistent way. This allows one to switch between the persistence technologies easily. It also allows you to code without worrying about catching exceptions that are specific to each of these technology.

### Programming & Frameworks Training

### ****32. Name the exceptions thrown by the Spring DAO classes.****

See the below diagram, it depicts all the Spring DAO classes in the hierarchical order.



### ****33.  Which classes are present in spring JDBC API?****

Classes present in JDBC API are as follows:

1. JdbcTemplate
2. SimpleJdbcTemplate
3. NamedParameterJdbcTemplate
4. SimpleJdbcInsert
5. SimpleJdbcCall

### ****34. What are the ways by which Hibernate can be accessed using Spring?****

There are two ways by which we can access Hibernate using Spring:

1. Inversion of Control with a Hibernate Template and Callback
2. Extending HibernateDAOSupport and Applying an AOP Interceptor node

### ****35. Name the types of transaction management that Spring supports.****

Two types of transaction management are supported by Spring. They are:

1. **Programmatic transaction management:** In this, the transaction is managed with the help of programming. It provides you extreme flexibility, but it is very difficult to maintain.
2. **Declarative transaction management:** In this, the transaction management is separated from the business code. Only annotations or XML based configurations are used to manage the transactions.

### ****36. What are the different ORM’s supported by Spring?****

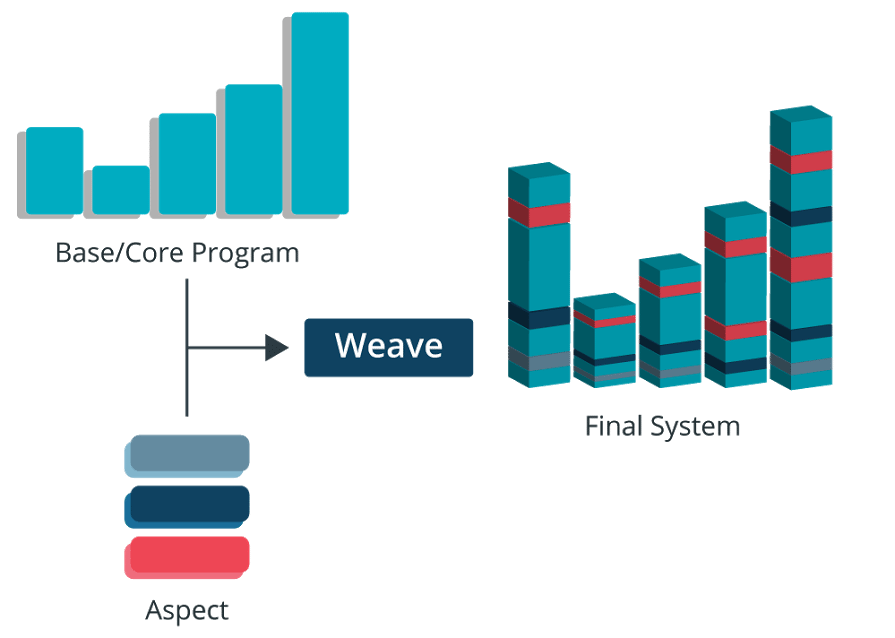
Different ORM’s supported by Spring are depicted via the below diagram:

The next section of Spring interview questions discusses on Spring AOP Interview Questions.

## **Aspect Oriented Programming (AOP) – Spring Interview Questions**

### ****37. Describe AOP.****

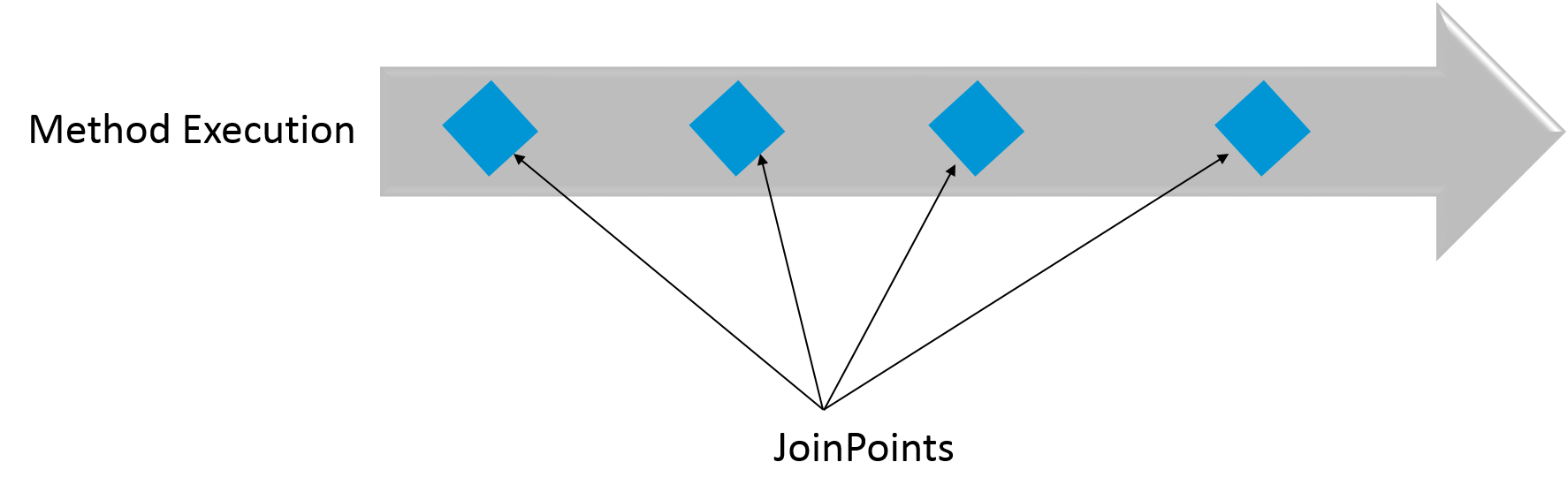
Aspect-oriented programming or AOP is a programming technique which allows programmers to modularize crosscutting concerns or behavior that cuts across the typical divisions of responsibility. Examples of cross-cutting concerns can be logging and transaction management. The core of AOP is an aspect. It encapsulates behaviors that can affect multiple classes into reusable modules.



### ****38. What do you mean by Aspect?****

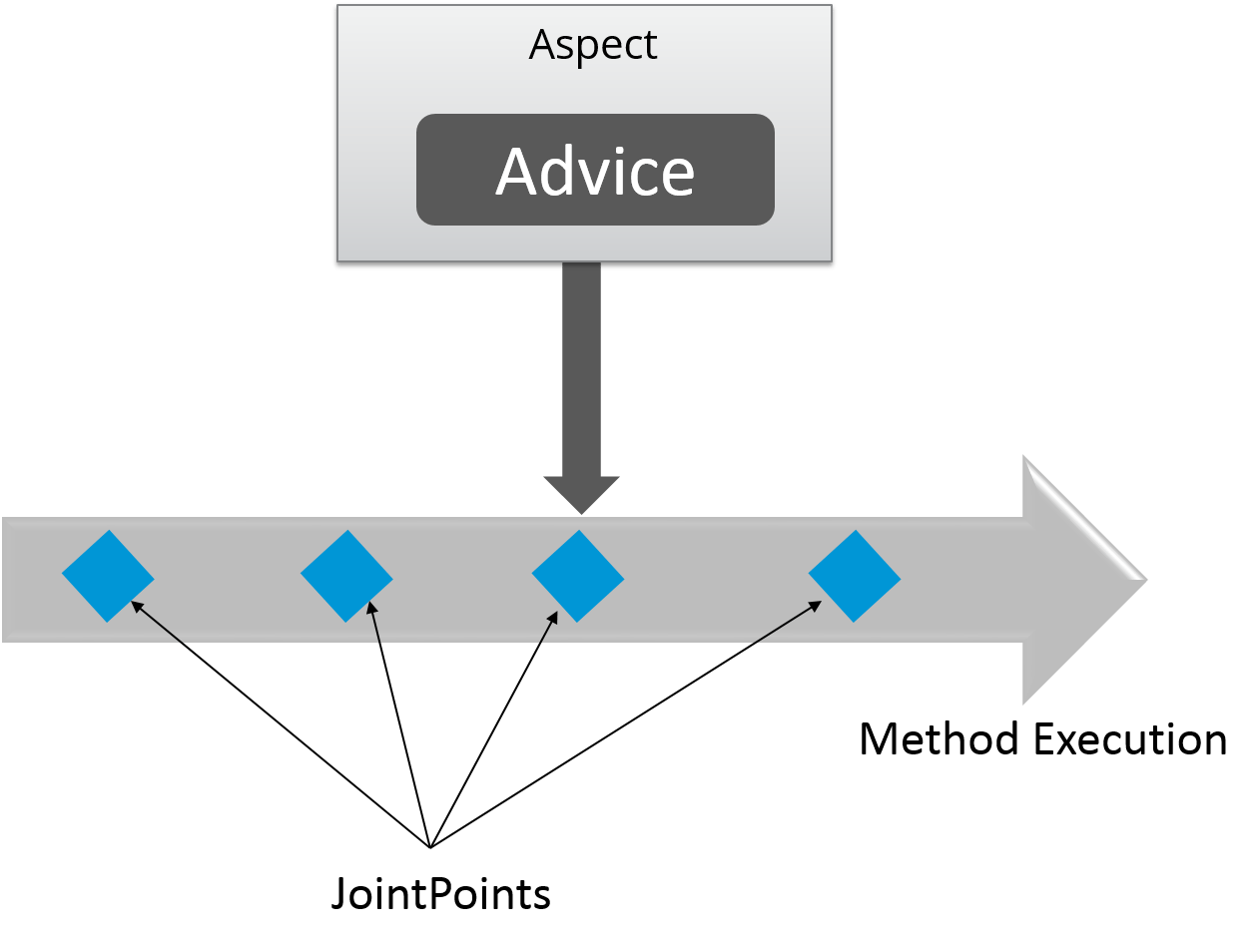
Aspect is a modularization of concern which cuts across multiple objects. Transaction management is a good example of a crosscutting concern in J2EE applications. Aspects are implemented using regular classes or regular classes annotated with the @Aspect annotation in Spring Framework.

### ****39. Explain JoinPoint.****

A point during the execution of a program is called JoinPoint, such as the execution of a method or the handling of an exception. In Spring AOP, a joinpoint always represents a method execution.

### ****40. What is an Advice?****

An Action taken by an aspect at a particular joinpoint is known as an Advice. Spring AOP uses an advice as an interceptor, maintaining a chain of interceptors “around” the join point.



### ****41. What are the different types of Advices?****

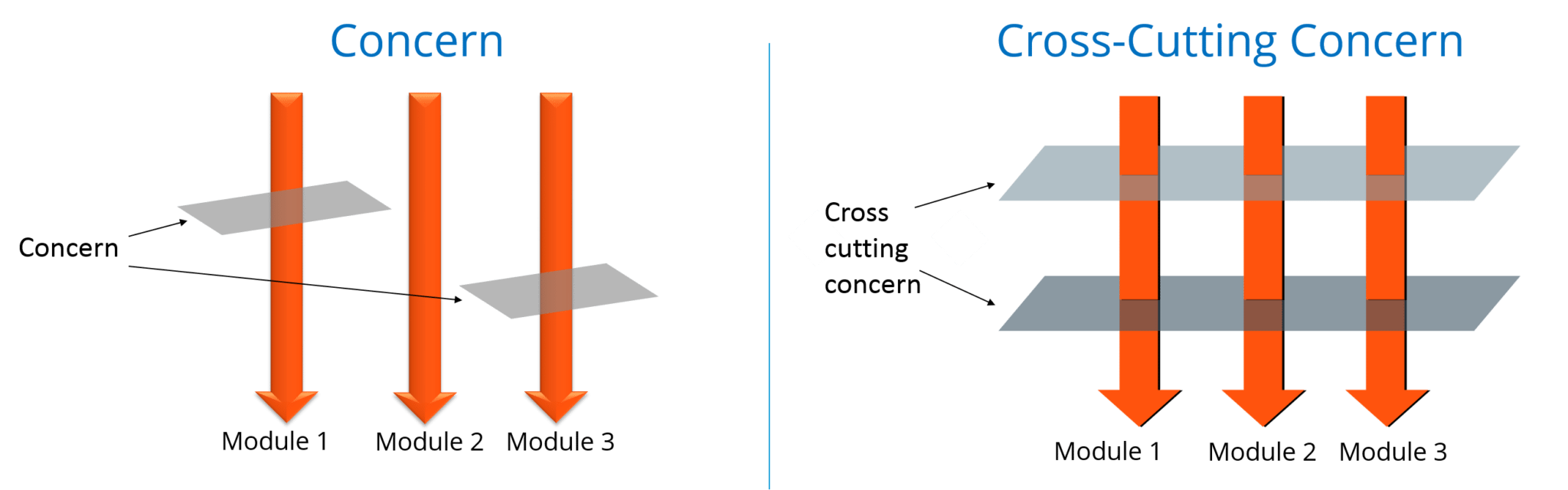
Different types of Advices in Spring AOP are:

1. **Before:** These types of advices execute before the joinpoint methods and are configured using **@Before**annotation mark.
2. **After returning:**These types of advices execute after the joinpoint methods completes executing normally and are configured using @AfterReturning annotation mark.
3. **After throwing:** These types of advices execute only if joinpoint method exits by throwing an exception and are configured using @AfterThrowing annotation mark.
4. **After (finally):** These types of advices execute after a joinpoint method, regardless of the method’s exit whether normally or exceptional return and are configured using @After annotation mark.
5. **Around:**These types of advices execute before and after a joinpoint and are configured using @Around annotation mark.

### ****42. Point out the difference between concern and cross-cutting concern in Spring AOP?****

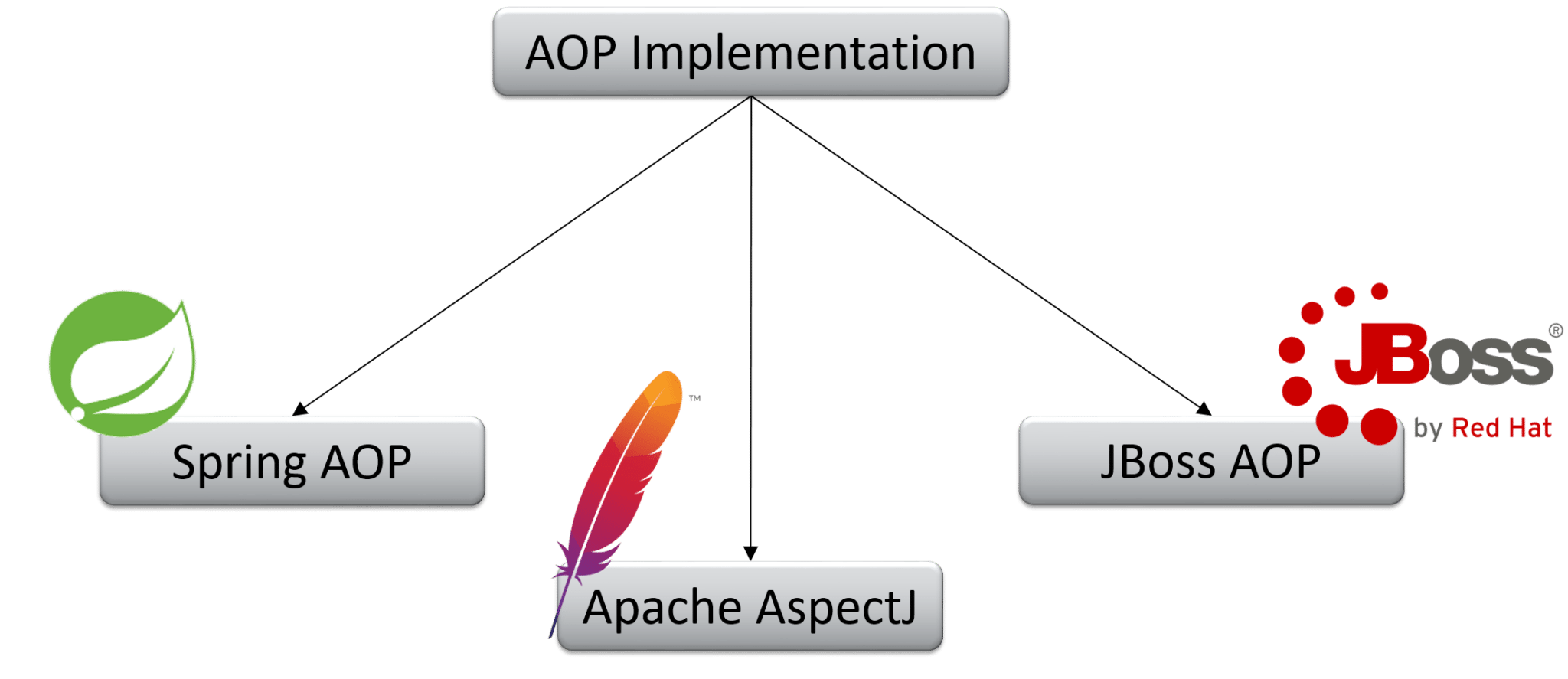
The concern is the behavior we want to have in a particular module of an application. It can be defined as a functionality we want to implement.

The cross-cutting concern is a concern which is applicable throughout the application. This affects the entire application. For example, logging, security and data transfer are the concerns needed in almost every module of an application, thus they are the cross-cutting concerns.



### ****43. What are the different AOP implementations?****

Different AOP implementations are depicted by the below diagram:



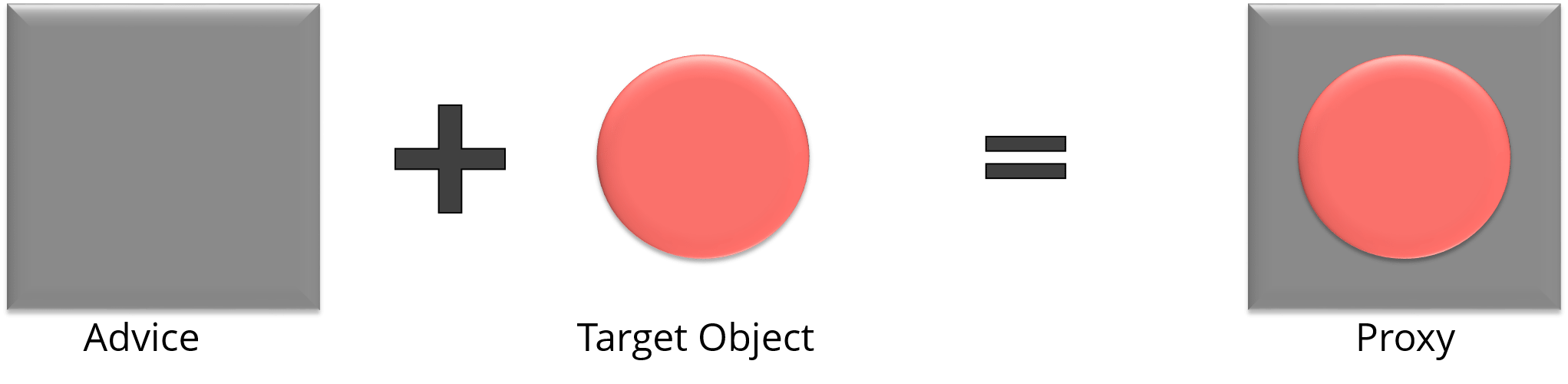
### ****44. What are the difference between Spring AOP and AspectJ AOP?****

#### Spring AOP vs AspectJ AOP

|  |  |
| --- | --- |
| **Spring AOP** | **AspectJ AOP** |
| Runtime weaving through proxy is done | Compile time weaving through AspectJ Java tools is done |
| It supports only method level PointCut | It suports field level Pointcuts |
| It is DTD based | It is schema based and Annotation configuration |

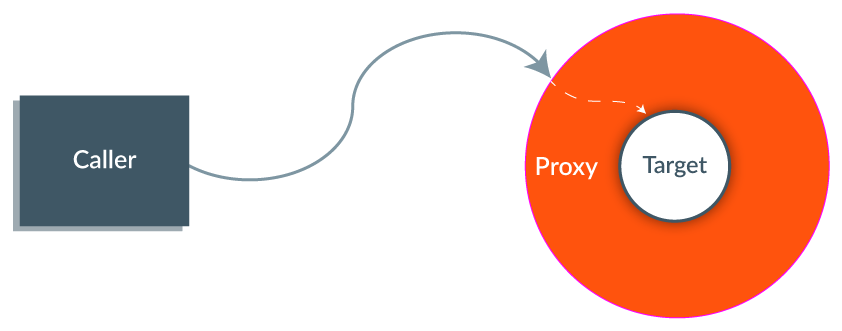
### ****45. What do you mean by Proxy in Spring Framework?****

An object which is created after applying advice to a target object is known as a Proxy. In case of client objects the target object and the proxy object are the same.



### ****46. In Spring, what is Weaving?****

The process of linking an aspect with other application types or objects to create an advised object is called Weaving. In Spring AOP, weaving is performed at runtime. Refer the below diagram:

The last section of Spring interview questions is on Spring MVC Interview Questions.

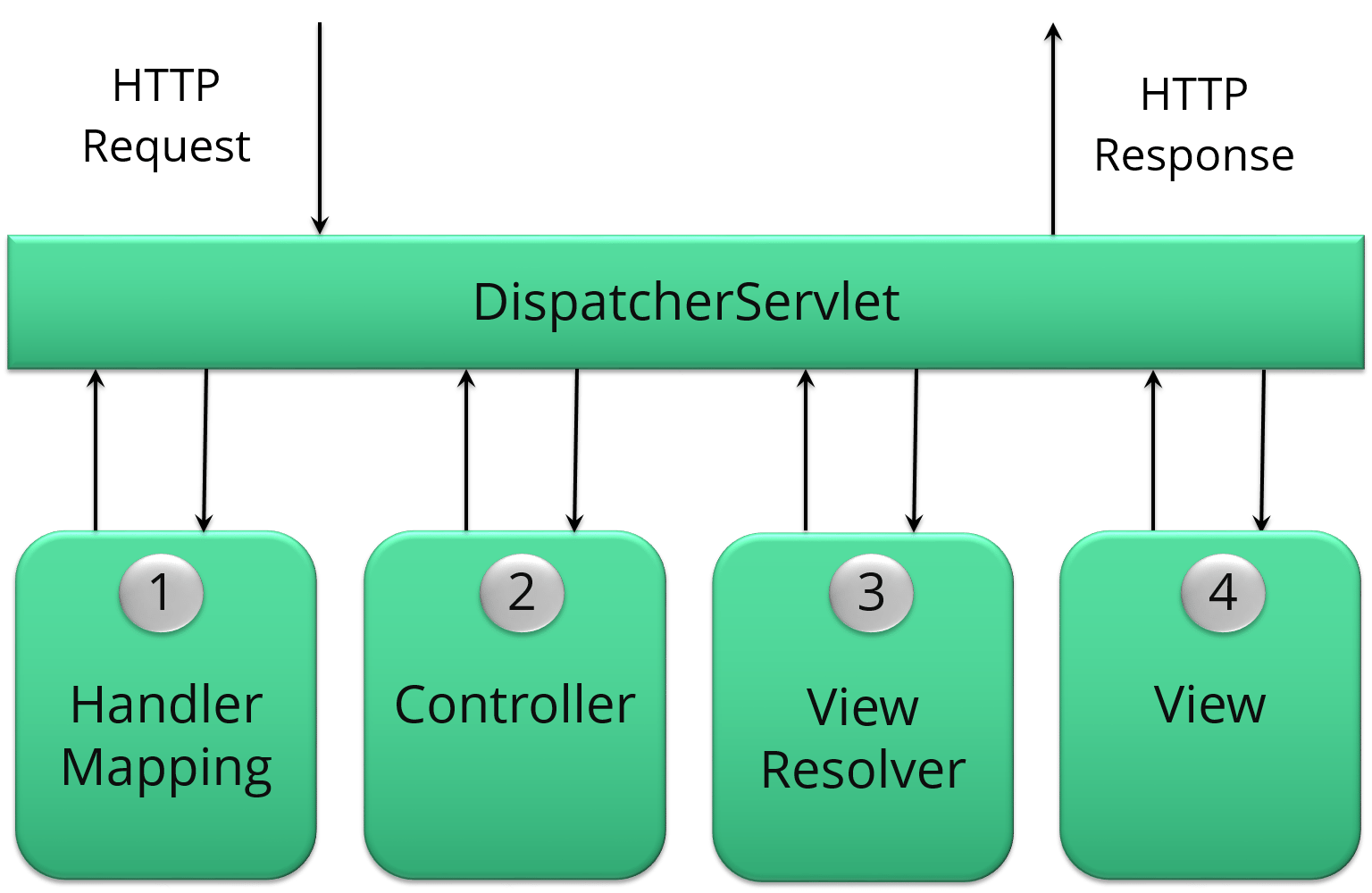
## **MVC (Model-View-Controller) – Spring Interview Questions**

### ****47. What do you mean by Spring MVC framework?****

The Spring web MVC framework provides model-view-controller architecture and ready to use components that are used to develop flexible and loosely coupled web applications. The MVC pattern helps in separating the different aspects of the application like input logic, business logic and UI logic, while providing a loose coupling between all these elements. [Get Started with Spring MVC](https://spring.io/guides/gs/serving-web-content/)

### ****48. Describe DispatcherServlet.****

The DispatcherServlet is the core of Spring Web MVC framework. It handles all the HTTP requests and responses. The DispatcherServlet receives the entry of handler mapping from the configuration file and forwards the request to the controller. The controller then returns an object of Model And View. The DispatcherServlet checks the entry of view resolver in the configuration file and calls the specified view component.



### ****49. Explain WebApplicationContext.****

The WebApplicationContext is an extension of the plain ApplicationContext. It has some extra features that are necessary for web applications. It differs from a normal ApplicationContext in terms of its capability of resolving themes and in deciding which servlet it is associated with.

### ****50. In Spring MVC framework, what is controller?****

Controllers provide access to the application behavior. These behaviors are generally defined through a service interface. Controllers interpret the user input and transform it into a model which is represented to the user by the view. In Spring, controller is implemented in a very abstract way. It also enables you to create a wide variety of controllers.

