**Interview Core Java**

**1. Difference between JDK, JRE and JVM?**

**Ans:** JDK, JRE, and JVM are all key components of the Java programming environment, but they serve different purposes:

**JDK (Java Development Kit) (Dev Tool+ JRE):**

* The JDK is a software development kit used by developers to develop and execute Java programs.
* Includes Development Tools: **The Java compiler (javac), debugger, and other development utilities**.
* JDK contains: JRE (which contains libraries and APIs) that developers use to build Java applications.

**JRE (Java Runtime Environment) (Runtime Env+ JVM):**

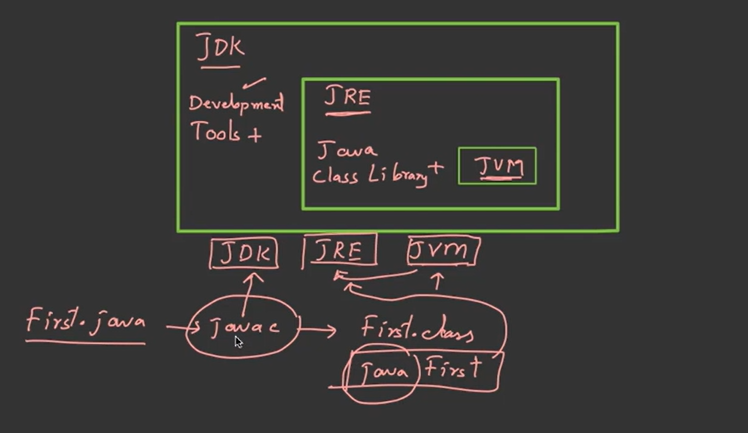
* The JRE is a runtime environment required to run Java applications on a computer.
* It includes the additional tools like class libraries, APIs and other supporting that developers use during the development process.
* These libraries and APIs provide functionality for a wide range of tasks, such as input/output operations, networking, database access, user interface development, and more.
* Sara class and interface issi me hota hai.
* It is an installation package that provides an environment to run the Java program or application on any machine.

**JVM (Java Virtual Machine) (Interprets Byte Code):**

* The JVM is an abstract computing machine that enables a computer to run Java bytecode.
* It interprets Java bytecode and translates it into machine-specific instructions that the underlying hardware can execute.
* It is responsible for loading, verifying, and executing the bytecode created in Java.
* JVM provides the runtime environment in which Java applications can be executed independently of the underlying hardware and operating system.

***In summary, the JDK is used by developers to write and compile Java code, the JRE is used by end-users to run Java applications, and the JVM is the runtime engine that executes Java bytecode on a computer.***

* JDK: Java Development Tool Kit: developments tools for compiling, debugging. + JRE.
* JRE: Java Runtime Environment: Java Class Library + JVM.
* JVM: Java Virtual Machine: This is actually executing the program.

**

*🡪First.java—>javac(JAVA Compiler is* part of JDK*)🡪 (this will give us a file called)🡪 First.class  
🡪for compilation we use javac—is development tool.  
🡪for execution we use java—it will start runtime environment.*

***JDK me hi javac hota hai jo usko .class file me convert karta hai , JRE usko runtime environment provide krta hai…***

***Que 2: What is Javac?***

***Ans :***  javac is the Java compiler, which is part of the Java Development Kit (JDK).

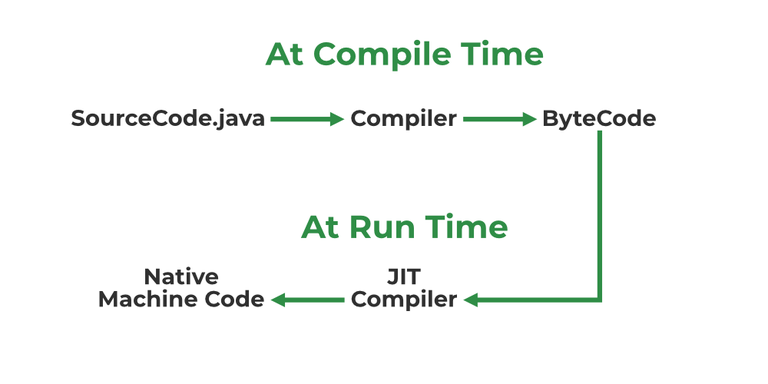
* It is a command-line tool used to compile Java source code files (.java files) into bytecode files (.class files) that can be executed by the Java Virtual Machine (JVM).
* After javac compiles Java source code, the resulting bytecode (.class files) can be executed by any JVM, regardless of the platform or operating system.

***In summary, javac is a tool provided by the JDK for compiling Java source code. Once compiled, the bytecode produced by javac can be executed by any JVM, which is part of the JRE.***

**Que 3: What is JIT?**

Ans: JIT stands for (Just-in-Time) compiler is a part of JRE(Java Runtime Environment).

* It is used for better performance of the Java applications during run-time.
* Source code is compiled with javac compiler to form bytecode
* Bytecode is further passed on to JVM.
* JIT is a part of JVM,
* JIT is responsible for compiling bytecode into native machine code at run time.



**Que 4: What is Class Loader?**

**Ans:**

* Class loader is the part of JRE (Java Runtime Environment), during the execution of the bytecode or created .class file class loader is responsible for dynamically loading the java classes and interfaces to JVM (Java Virtual Machine).
* Because of class loaders Java run time system does not need to know about files and file systems.
* This includes both system classes (e.g., core Java libraries) and user-defined classes.
* The Class Loader searches for classes in directories and JAR (Java Archive) files specified in the classpath environment variable or command-line options. It locates and loads the bytecode of classes from these locations.

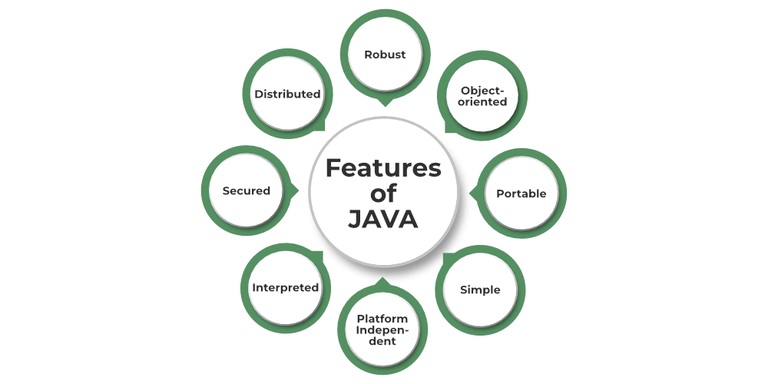
**Que: What is Delegation Model Class Loader?**

**Ans:** The delegation model, also known as the delegation hierarchy or class loading delegation, is a fundamental concept in Java's class loading mechanism.

* It defines the order in which Java Class Loaders attempt to load classes and resources, and it operates based on a hierarchical structure.
* Each library follows a "parent-first" strategy, meaning it asks its parent library to provide the book before trying to find it on its own shelves.
* Example: Hierarchy: Imagine a hierarchy like a family tree, where there's a root (the oldest ancestor) and multiple levels of descendants.
* In Java's class loading, the hierarchy starts with the Bootstrap Class Loader at the top (root), followed by other Class Loaders as its descendants, forming a tree-like structure.

**Que 5: What are top features of Java?**

**Ans:** There are many features in Java that makes it better than any other language some of them are mentioned below:



* Simple: Java is quite simple to understand and the syntax
* Platform Independent: Java is platform independent means we can run the same program in any software and hardware and will get the same result.
* Interpreted: Java is interpreted as well as a compiler-based language.
* Robust: features like Garbage collection, exception handling, etc. that make the language robust.
* Object-Oriented: Java is an object-oriented language that supports the concepts of class, objects, four pillars of OOPS, etc.
* Secured: As we can directly share an application with the user without sharing the actual program makes Java a secure language.
* High Performance: faster than other traditional interpreted programming languages.
* Dynamic: supports dynamic loading of classes and interfaces.
* Distributed: feature of Java makes us able to access files by calling the methods from any machine connected.
* Multithreaded: deal with multiple tasks at once by defining multiple threads Architecture Neutral: it is not dependent on the architecture.

**Que 6: Explain public static void main(String args[]) in Java?**

**Ans:** In Java, the public static void main(String args[]) method is the entry point for a Java application. When you run a Java program, the Java Virtual Machine (JVM) starts executing the code from this method.

JVM looks for the main method when it starts execution.

* ***Public: (*Access Specifier*)*** It is responsible for making the main function globally available. It allows the JVM to access and execute the main method.
* ***Static: (Keyword)*** We cannot use the class without creating an object, so if you want to use the class without creating an object then you must use ***static.***

*The static keyword indicates that the main method belongs to the class itself, rather than to any particular instance of the class. This means that you can invoke the main method without creating an object of the class.*

***Java First 🡪 it will call First. Main()🡪 it will create an object🡪 If you write static, it won’t create🡪 If you run without static then it will show error (not able to find class).***

**Sub Que: What will happen if you don’t declare main as static?**

**Ans:** We can declare the main method without using static and without getting any errors. But, the main method will not be treated as the entry point to the application or the program.

* ***Void: (Return Type)*** It specifies that main method doesn’t return anything.

In other words, the main method is not expected to produce any result that needs to be returned to the caller.

*As the main() terminate, java program also terminates. It doesn’t make any sense to get return from main. JVM Kya krega us return ka bcz program is over.*

* ***main()*** is a method which is entry point of java program.

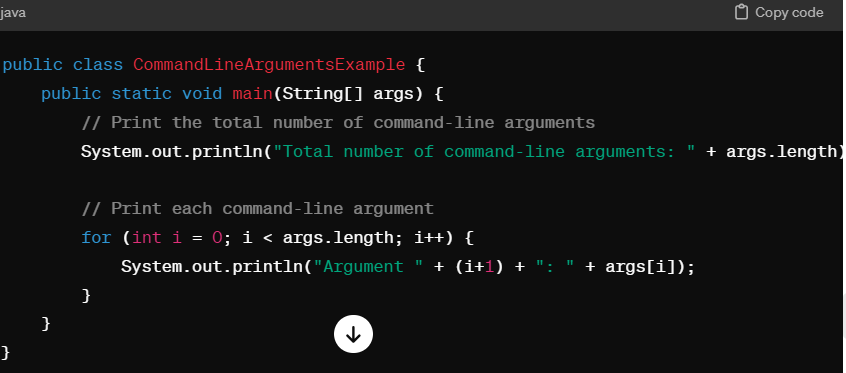
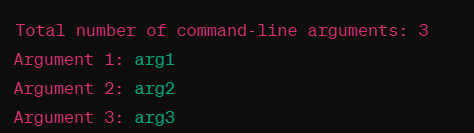
It is the name of the method. It is predefined in Java as the starting point for the execution of a program.

* ***String arg[]***  It’s a Java command line arguments.

This is the parameter of the main method. It is an array of strings (String[]) named args. This parameter allows you to pass command-line arguments to your Java program. When you run a Java program from the command line, you can provide additional arguments after the name of the class to customize its behavior. These arguments are then stored in the args array, and you can access them within the main method.

**Que 7: What is command Line Arguments?**

**Ans:** Command-line arguments are values provided to a program when it is executed from the command line or terminal. They allow users to customize the behavior of a program by passing additional information to it during runtime. In Java, command-line arguments are typically accessed through the String[] args parameter of the main method.

🡪 

Compile the code and run the class file by:

***java CommandLineArguments arg1 arg2 arg3***

**Que 8:**

* **What are Packages in Java?**
* **Why are Packages used?**
* **Advantages of Packages in Java?**
* **How many types of packages are there in Java?**

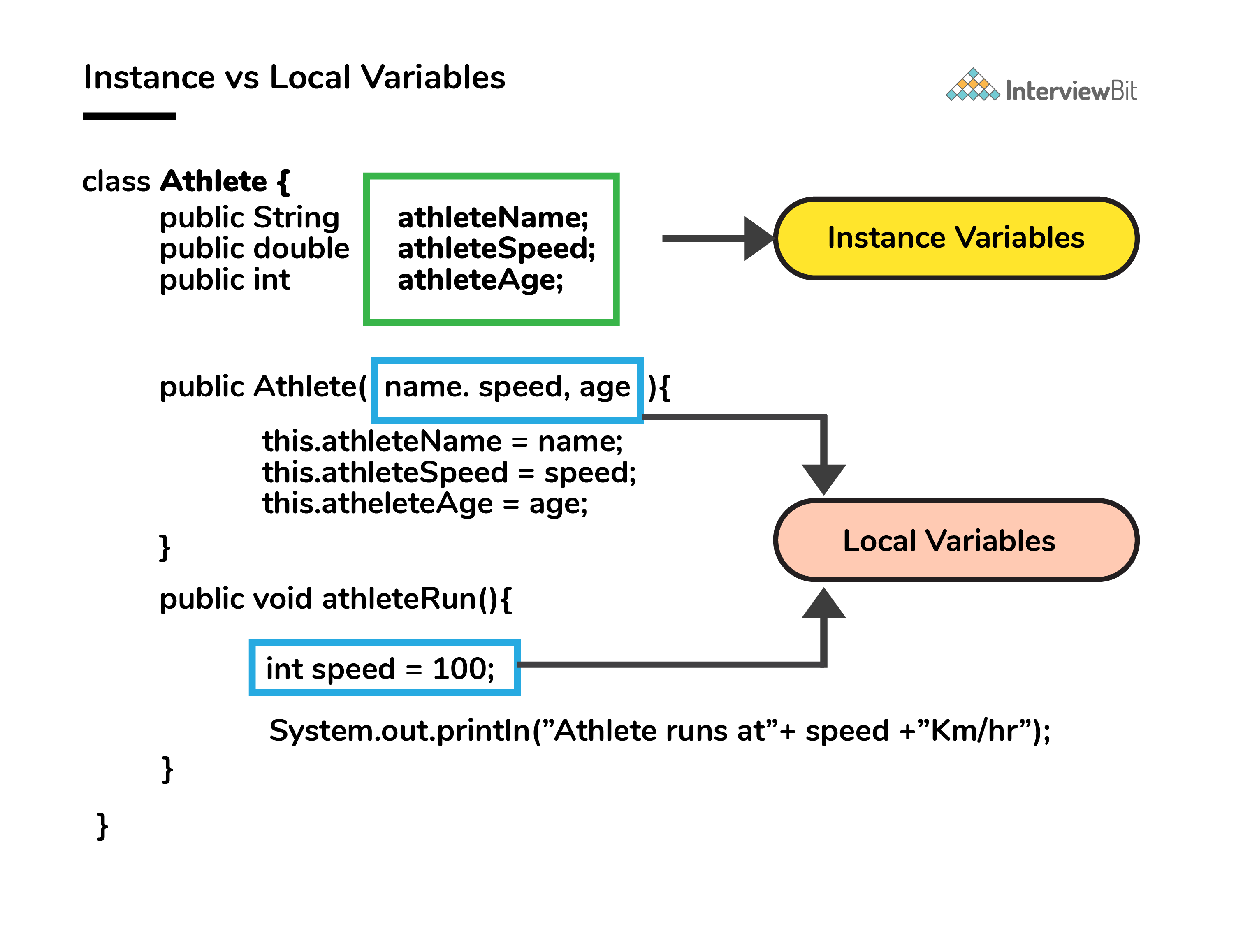
**Ans:**

* Packages in Java can be defined as the grouping of related types of classes, interfaces, etc providing access to protection and namespace management.
* Packages are used in Java in order to prevent naming conflicts, control access, and make searching/locating and usage of classes, interfaces, etc easier.
* There are two types of packages in Java:

User-defined packages.

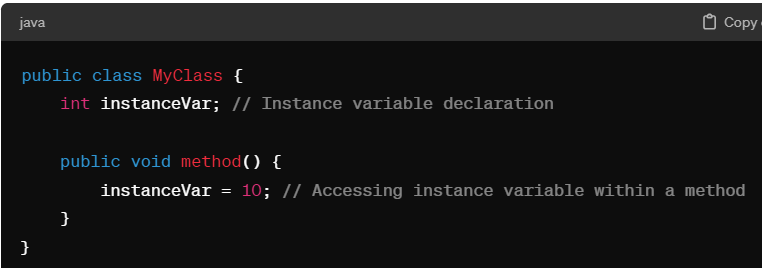
Build-In packages.

**Que 9: Difference between Local variable and Instance Variables?**



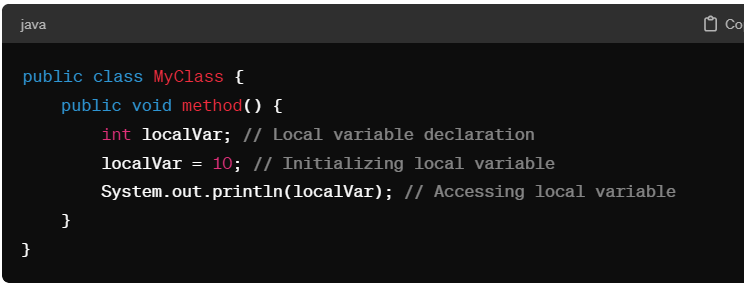
**Ans: Instance Variables:**

* Instance variables are declared within a class but outside of any method, constructor, or block.
* They are also known as member variables or fields.
* Each instance of the class (i.e., each object) has its own copy of instance variables.
* Instance variables are initialized when an object of the class is created and are accessible throughout the entire lifetime of the object.
* Instance variables are associated with objects and exist as long as the object exists.
* When we haven’t initialized the instance variables then the compiler initializes them with default values.



**Local Variables:**

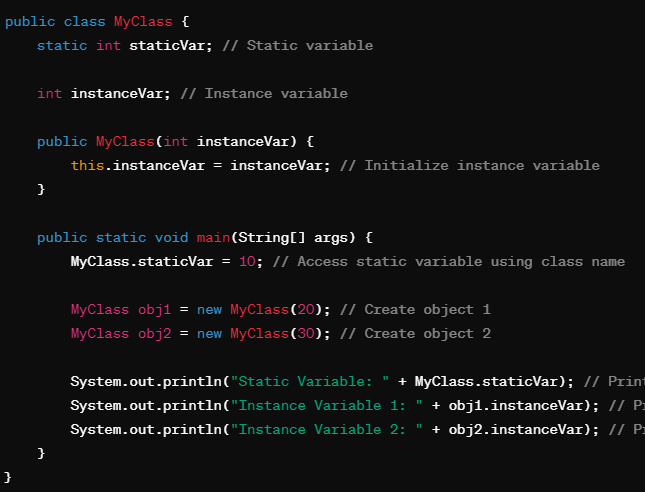
* Local variables are declared within a method, constructor, or block and are only accessible within that scope.
* They are created when the method, constructor, or block is entered and destroyed when it exits.
* Local variables must be explicitly initialized before they are used, either by assigning a value or by reading from an input source.
* They do not have default values and must be initialized before use to avoid compilation errors.



**Que 10:** **What is class Variable (static Variable)?**

**Ans:**

* In Java, a class variable (also known as a static variable) is a variable that is declared within a class but outside of any method, constructor, or block.
* Class variables are declared with the static keyword, and they are shared by all instances (objects) of the class as well as by the class itself.
* No matter how many objects are derived from a class, each class variable would only exist once.
* It's important to note that static members are not tied to any particular instance of the class and can be accessed even if no objects of the class have been created.
* A static variable is associated with the class itself rather than with any specific instance of the class. It is shared among all instances of the class and can be accessed using the class name directly.



**Que 11: How Instance Variable is different from Static Variable?  
Ans:**  Key differences between instance variables and static variables:

**Scope:**

* Instance variables have instance scope, meaning they are specific to each object (instance) of the class.
* These variables can have distinct values among several objects.
* Static variables have class scope, meaning they are shared by all instances of the class.

**Memory Allocation:**

* Instance variables are allocated memory each time an object is created from the class. Each object has its own copy of instance variables.
* Static variables are allocated memory only once when the class is loaded into memory. They exist throughout the lifetime of the program and are shared among all instances of the class.

**Access:**

* Instance variables are accessed through object references (this keyword), and their values can vary from object to object.
* A static variable is associated with the class itself rather than with any specific instance of the class. It is shared among all instances of the class and can be accessed using the class name directly.

**Initialization:**

* Instance variables are initialized when an object is created, typically in constructors or with initializers.
* Static variables are initialized when the class is loaded into memory, typically with static initializers or directly at the point of declaration.

**Usage:**

* Instance variables are used to store unique data for each object and represent the state of individual objects.
* Static variables are used for data that is shared among all instances of the class, such as constants, global counters, or shared resources.

**Que : What is the difference between System.out, System.err, and System.in?**

**Ans:**

* **System.out –**  It is a PrintStream that is used for writing characters or can be said it can output the data we want to write on the Command Line Interface console/terminal.
* **System.err –** It is used to display error messages.

System.err have many similarities both of them have quite a lot of difference also, let us check them.

| **System.out** | **System.err** |
| --- | --- |
| It will print to the standard out of the system. | It will print to the standard error. |
| It is mostly used to display results on the console. | It is mostly used to output error texts. |
| It gives output on the console with the default(black) color. | It also gives output on the console but most of the IDEs give it a red color to differentiate. |

* **System.in –** It is an InputStream used to read input from the terminal Window. We can’t use the System.in directly so we use Scanner class for taking input with the system.in.

**Que: Difference between Print and Println?**

**Ans:** In Java, print and println are methods used to output text to the console. They are both part of the System.out object, which is an instance of the PrintStream class.

The main difference between them lies in how they handle newline characters:

**print:**

* The print method is used to display text to the console without advancing to the next line.
* After printing the specified text, the cursor remains on the same line.

**println:**

* The println method is used to display text to the console and then move the cursor to the next line.
* After printing the specified text, the cursor moves to the beginning of the next line.

**Printf:**

* It allows you to specify a format string containing placeholders for variables, followed by the values to be substituted into those placeholders.
* This method is part of the PrintStream class, like print and println.



**Que: Define System.out.println().**

**Ans:** System.out.println() is used to print the message on the console. System - It is a class present in java.lang package. Out is the static variable of type PrintStream class present in the System class. println() is the method present in the PrintStream class.

So if we justify the statement, then we can say that if we want to print anything on the console then we need to call the println() method that was present in PrintStream class. And we can call this using the output object that is present in the System class.

**Que: What is scanner class?**

**Ans:**

* The Scanner class in Java is a part of the java.util package and is used for reading input of different types from various sources, such as the keyboard, files, or strings.
* It provides methods to parse primitive types and strings from the input stream.
* Reading Input: The Scanner class provides methods like nextInt(), nextDouble(), nextLine(), etc., to read different types of input from the source.

**nextInt():**

* The nextInt() method reads the next token from the input source and attempts to interpret it as an integer.
* It skips any leading whitespace and tries to read characters until it encounters a non-digit character or the end of the input.
* If the token can be interpreted as an integer, it returns the integer value. Otherwise, it throws an InputMismatchException.

**nextDouble():**

* The nextDouble() method reads the next token from the input source and attempts to interpret it as a floating-point number.
* nextDouble(), but it expects a floating-point number instead of an integer.
* If the token can be interpreted as a double, it returns the double value. Otherwise, it throws an InputMismatchException.

**nextLine()**

* The nextLine() method reads the next line of input from the input source, including any whitespace characters.
* It reads characters until it encounters a newline character ('\n') or the end of the input.
* It returns the input line as a string, excluding the newline character at the end.

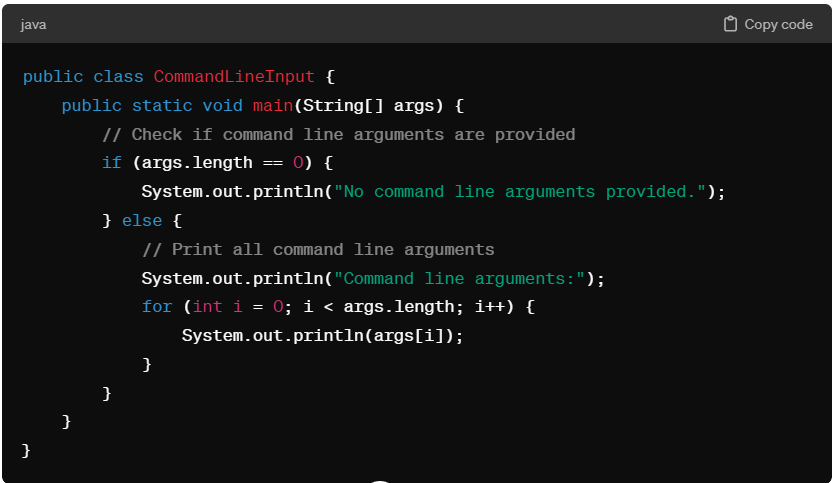
**Que: How many ways you can take input from the console?**

**Ans:** There are two methods to take input from the console in Java mentioned below:

* Using Command line argument
* Using Buffered Reader Class
* Using Console Class
* Using Scanner Class

**1. Using Command line argument:**

In Java, you can take input from the command line arguments using the args parameter in the main method. The args parameter is an array of strings that contains the command line arguments passed to the Java program when it is executed.



In this example:

* The main method accepts an array of strings args, which represents the command line arguments.
* We first check if any command line arguments are provided by checking the length of the args array.
* If no arguments are provided, we print a message indicating that no arguments were provided.
* If arguments are provided, we loop through the args array and print each argument.

To run this program with command line arguments, you can compile the Java source file and then execute the compiled class file with the arguments separated by spaces. For example:



**2. Using Buffered Reader Class:**

**3. Using Console Class:**

In Java, the `Console` class provides methods for reading input from the console and writing output to the console. It was introduced in Java 6 as part of the `java.io` package.

The `Console` class is primarily used for interacting with the user when running applications from the command line. It allows you to read input from the user without resorting to `System.in` and `Scanner`, and also provides a convenient way to output text to the console.

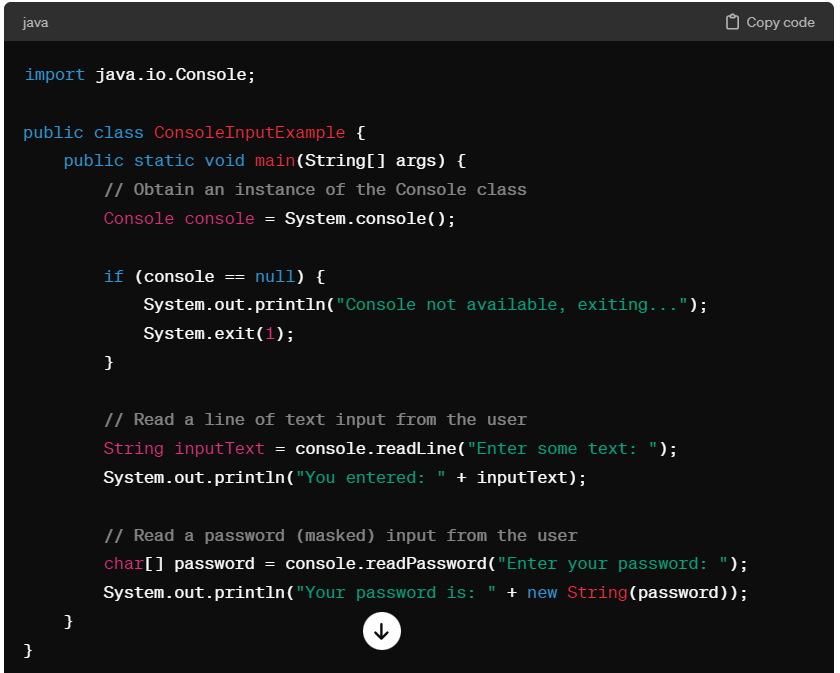
Some of the commonly used methods of the `Console` class include:

* **`readLine(String fmt, Object... args)`:** Reads a line of text from the console, optionally prompting the user with the specified format string and arguments.
* **`readPassword(String fmt, Object... args)`**: Reads a password from the console without echoing characters to the screen, providing a level of security.
* **`format(String fmt, Object... args)`:** Writes a formatted string to the console.
* **`printf(String fmt, Object... args)`:** A convenience method similar to `format`, but with a name more familiar to users of C-style formatted output.

It's important to note that the `Console` class has limitations. It may return `null` from `System.console()` if the JVM is not connected to a console (for example, when running from an IDE or as a background process). In such cases, you would need to resort to other input/output mechanisms like `Scanner` or standard input/output streams (`System.in` and `System.out`).

To take input in Java using the Console class, you can follow these steps:

* Obtain an instance of the Console class.
* Use the methods provided by the Console class to read input from the user.



Note:

1. The System.console() method returns null if the JVM is not connected to a console, such as when running from an IDE or as a background process. In such cases, you may want to handle this scenario gracefully.

2. The readPassword method does not echo characters to the console for security reasons. It returns a char[] array containing the characters entered by the user. You should be careful not to leave this array lying around in memory to minimize the risk of exposing sensitive information.

**Que: What is This Keyword?**

**Ans:** In Java, "**this" is a keyword that refers to the current instance of the class in which it appears**. It is primarily used within instance methods or constructors to refer to the current object.

When there is a local variable with the same name as an instance variable, "this" can be used to distinguish between the two.

public class MyClass {

private int x; 🡪 Instance variable, declared within class but outside any method or constructor.

public void setX(int x) { 🡪 is a method , It's a member function of the MyClass class.

🡪Integer x is a parameter

this.x = x; // "this" refers to the instance variable x

} this.x wala x refers to instance variable

second x refers to parameter passes to the setX method.

}

this ka mtlb hai iss object ke liye jo x upr class se liye hai wo wale x ki value ye parameter wala x hoga.

Remarks:

**The this keyword is used to distinguish between the instance variables and the parameters passed to the constructor. It indicates that we are assigning the values passed as parameters to the instance variables of the current object.**

**Que: Why is Java a platform independent language?**

**Ans**: Java is platform-independent language and was developed so that it does not depend on any hardware or software because it compiles source code into bytecode, which is executed by the Java Virtual Machine (JVM). The same bytecode can run on any platform with a compatible JVM, allowing Java programs to be written once and run anywhere without modification.

**Que: Why is Java not a pure object oriented language?**

**Que:** **Can java be said to be the complete object-oriented programming language?**

**Ans:** Java is not considered a pure object-oriented language because it supports primitive data types (such as int, double, boolean) that are not objects. These primitive types are not instances of classes and do not inherit from a common object class like other objects in Java. Additionally, Java allows procedural programming constructs (like static methods and variables) that are not purely based on object-oriented principles.

**Que: How is Java different from C++?**

**Ans:**

* C++ is only a compiled language, whereas Java is compiled as well as an interpreted language.
* Java programs are machine-independent whereas a c++ program can run only in the machine in which it is compiled.
* C++ allows users to use pointers in the program. Whereas java doesn’t allow it. Java internally uses pointers.
* C++ supports the concept of Multiple inheritances whereas Java doesn't support this. And it is due to avoiding the complexity of name ambiguity that causes the diamond problem.

**Que: Pointers are used in C/ C++. Why does Java not make use of pointers?**

**Ans:** 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.

**Que: What are the default values assigned to variables and instances in java?**

**Ans:** There are no default values assigned to the variables in java. We need to initialize the value before using it. Otherwise, it will throw a compilation error of (Variable might not be initialized).

But for instance, if we create the object, then the default value will be initialized by the default constructor depending on the data type.

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

**Ans:**

**1. equals() Method:**

* equals() is a method defined in the Object class, which is the root class for all Java objects. It is meant for comparing the content or value equality of objects.
* When you call equals() on two objects, it compares the actual contents of the objects (i.e., the values they hold) to determine if they are equal.
* For most Java classes, the default implementation of equals() checks for reference equality (i.e., whether two object references point to the same memory location). However, many classes override the equals() method to provide their own custom logic for content-based comparison.
* Example: String class overrides equals() to compare the actual content of two String objects, not just their memory addresses.

**2.Equality Operator (==):**

* The equality operator (==) is a binary operator used in Java to compare the references of objects, not their content.
* When you use == to compare two object references, it checks if they refer to the exact same object in memory (i.e., the same memory address).
* If two variables containing object references use == and both references point to the same object, the result is true. Otherwise, it is false.
* Example: String objects created with the same content may not have the same memory address but can still be equal in content. Using == on such String objects may yield false, while using equals() would yield true if they have the same content.

**Que: How is an infinite loop declared in Java?**

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

***for(;;) while(true) do***

***{ { {***

***Business Logic Business Logic Business Logic***

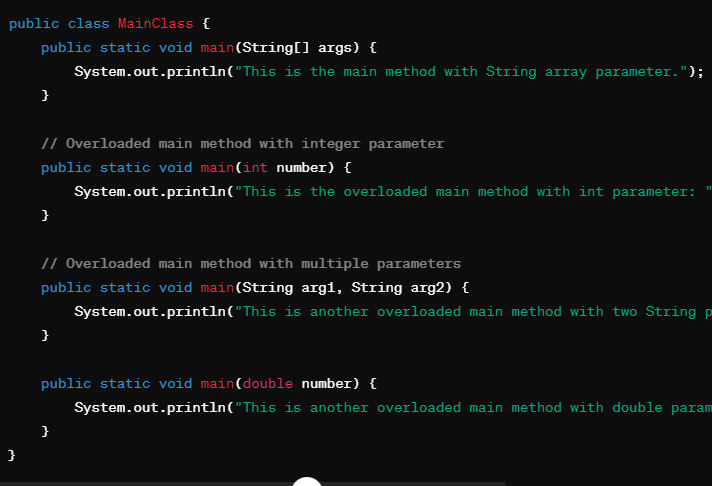
***Any break logic Any break logic Any break logic***

***} } }***

***while(true);***

**Que: Can the main method be Overloaded?**

**Ans:** Yes, the main method in Java can be overloaded, just like any other method in a Java class. Method overloading refers to the ability to have multiple methods with the same name but different parameter lists within the same class.

******

The MainClass contains multiple overloaded main methods, each with different parameter types. When you run the Java program, the JVM looks for the standard main method with the signature public static void main(String[] args) to start the program.

**Que: What are the two type of memory used in Java?**

**Que: Difference between Heap and Stack Memory in java. And how java utilizes them.**

**Ans:**

**Heap Memory:**

* Heap memory is the area of memory where objects and their instance variables are allocated.
* All objects created in Java are stored in the heap memory, regardless of whether they are created using the new keyword or as part of automatic memory management (e.g., string literals).
* The heap memory is managed by the Java Virtual Machine (JVM) and is dynamically allocated and deallocated as needed during program execution.
* Garbage collection is the process by which unused objects in the heap memory are identified and removed to free up space for new objects.

**Stack Memory:**

* Stack memory is used for storing method frames and local variables for each method invocation or function call in Java.
* Each time a method is called, a new frame is created on the stack to store local variables, method arguments, and return addresses.
* The stack memory operates in a Last-In-First-Out (LIFO) manner, where the most recently called method is at the top of the stack and is popped off when the method execution completes.
* Primitive data types, references to objects, and method call information are typically stored in the stack memory.
* Stack memory is faster to access compared to heap memory but has limited size and is shared among all threads in a Java application.

**Utilizes this memory as -**

When we write a java program then all the variables, methods, etc are stored in the stack memory.

And when we create any object in the java program then that object was created in the heap memory. And it was referenced from the stack memory.

***class Main {***

***public void printArray(int[] array){***

***for(int i : array)***

***System.out.println(i);***

***}***

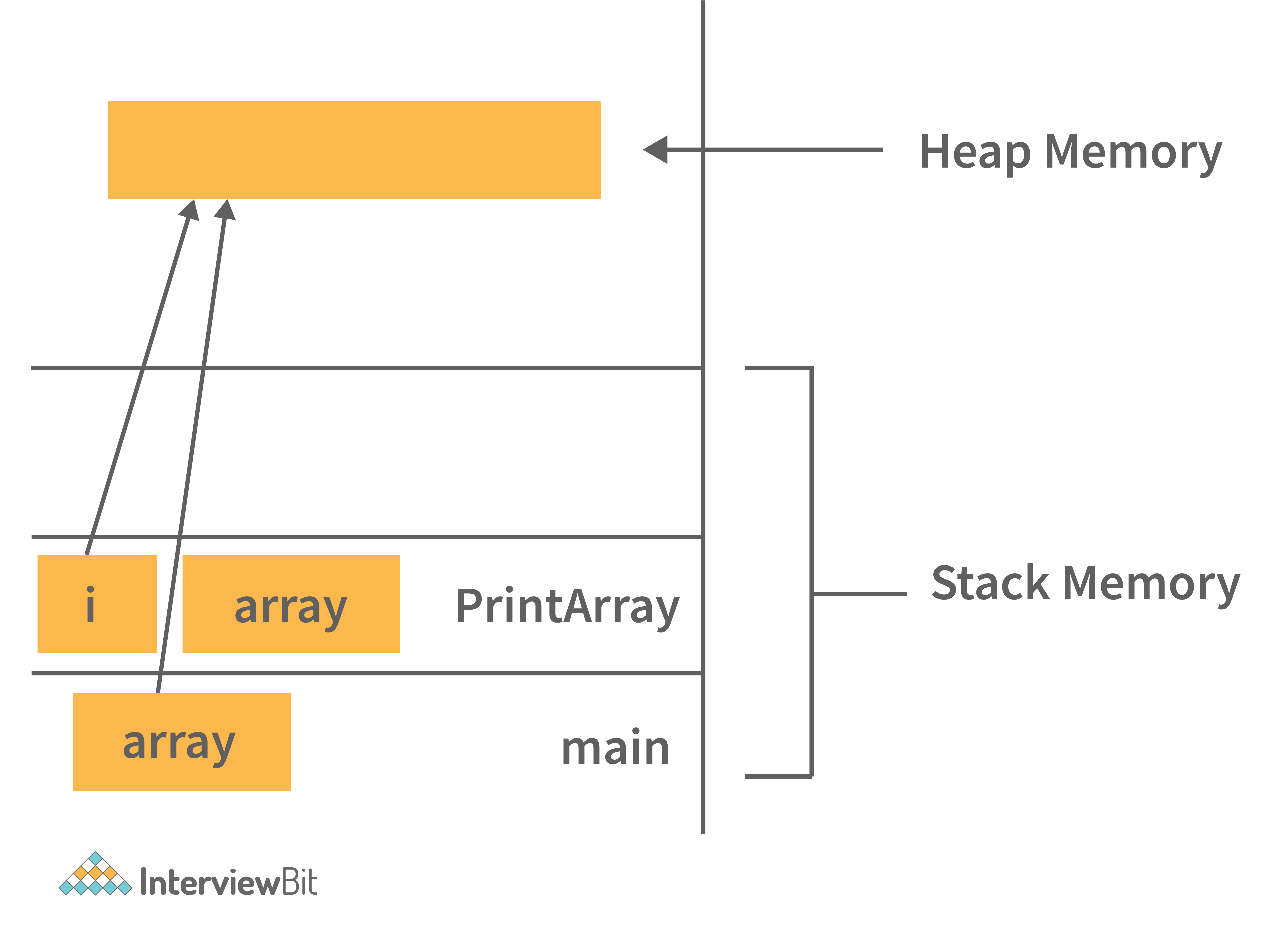
***public static void main(String args[]) {***

***int[] array = new int[10];***

***printArray(array);***

***}***

***}***



* Main and PrintArray is the method that will be available in the stack area and as well as the variables declared that will also be in the stack area.
* And the Object (Integer Array of size 10) we have created, will be available in the Heap area because that space will be allocated to the program during runtime.

**Que: What is Garbage Collection in Java?**

**Ans**: Garbage collection in Java is an automatic memory management process where the Java Virtual Machine (JVM) automatically identifies and removes unused objects from memory to free up space. The main purpose of garbage collection is to reclaim memory occupied by objects that are no longer needed or referenced by the program, preventing memory leaks and improving overall memory efficiency.

**Que: What part of memory - Stack or Heap - is cleaned in garbage collection process?**

**Ans: Heap**

Garbage collection primarily targets the heap memory, where Java objects and their instance variables are allocated. It identifies and removes unreferenced objects (objects without any active references) from the heap.

**Que: How does Garbage collection works in Java?**

**Ans:** Mark and Sweep Algorithm: The most commonly used garbage collection algorithm in Java is the Mark and Sweep algorithm. It involves two main phases:

* **Mark Phase:** During this phase, the garbage collector traverses the heap memory starting from the root objects (objects directly accessible from the stack or static variables) and marks all reachable objects as live.
* **Sweep Phase**: In this phase, the garbage collector sweeps through the heap and deallocates memory for objects that were not marked as live (i.e., unreachable objects). It then compacts the remaining memory to reduce fragmentation.

**Que: What are shallow copy and deep copy in java?**

**Ans:** In Java, shallow copy and deep copy are two different ways of copying objects, each with its own implications for how the object's contents are duplicated.

**1. Shallow Copy:** The shallow copy only creates a new reference and points to the same object.

***Rectangle obj2 = obj1;***

* Shallow copy refers to creating a new object that copies all the fields of the original object. However, if the fields themselves are references to other objects, the shallow copy will only copy the references, not the actual objects they point to.
* In other words, the copied object and the original object will share references to the same objects in memory for fields that are reference types (objects).
* Shallow copy is achieved using methods like Object.clone() or by implementing the Cloneable interface and overriding the clone() method.

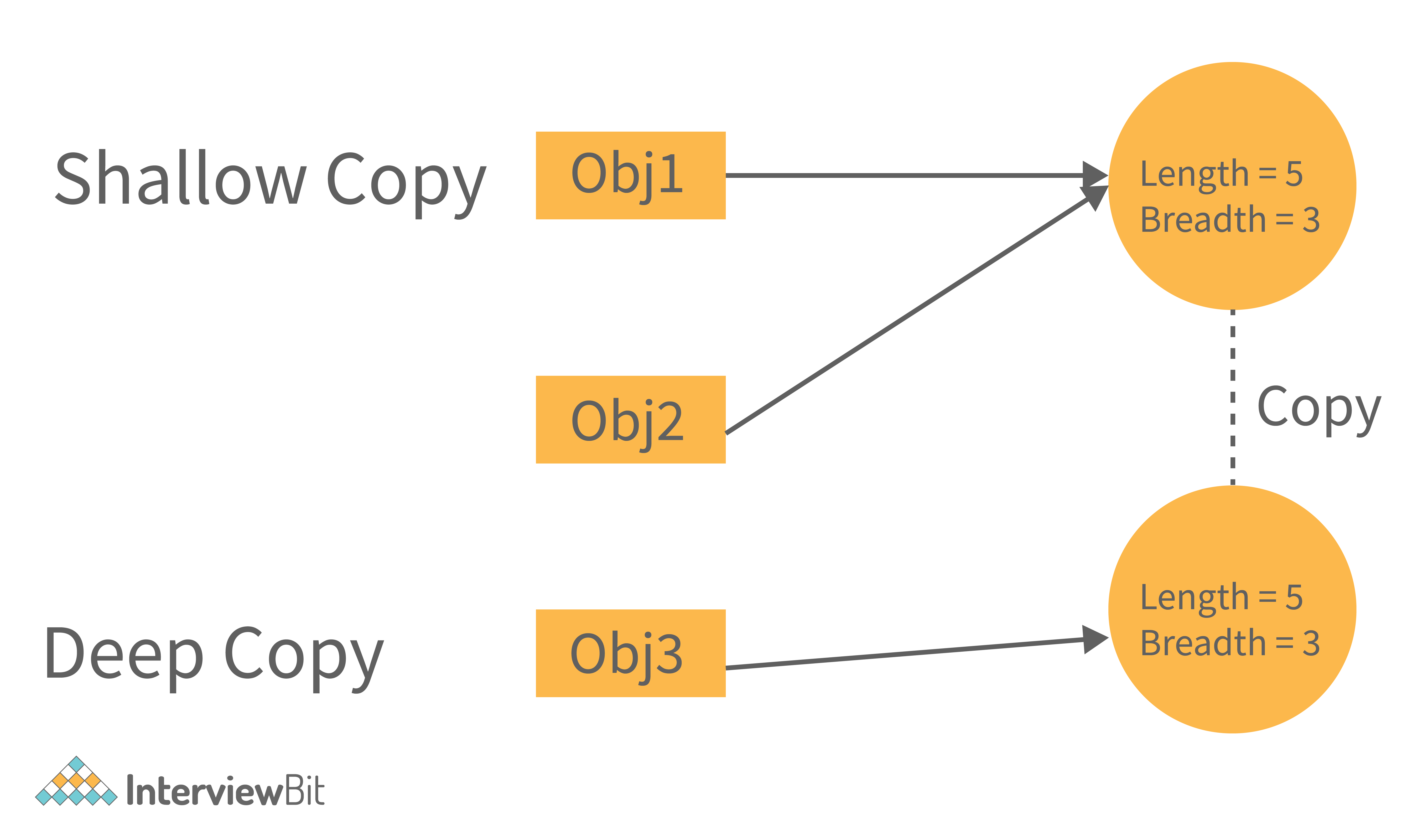
**2. Deep Copy:** In a deep copy, we create a new object and copy the old object value to the new object.

***Rectangle obj3 = new Rectangle();***

***Obj3.length = obj1.length;***

***Obj3.breadth = obj1.breadth;***

* Deep copy refers to creating a new object that duplicates not only the fields of the original object but also recursively copies all referenced objects. This ensures that the copied object and the original object have independent copies of all objects they reference.
* Deep copy is typically implemented using custom methods or libraries that traverse the object graph and create copies of all nested objects.
* Deep copy is more complex and slower than shallow copy but ensures that changes to the copied object's state do not affect the original object or vice versa.



**Que: What happens if there are multiple main methods inside one class in Java?**

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

**Que: What happens if the static modifier is not included in the main method signature in Java?**

**Ans:** 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.

**Que: Que: What are immutable class and how to create it?**

**Ans:** An immutable class in Java is a class whose instances cannot be modified after they are created. This is achieved by making the class final, ensuring that its state cannot be changed once initialized, and making its fields private and final to prevent direct modification. We cannot create the subclass of immutable created.

**public final class ImmutableClass {**

**// Private final fields**

**private final int intValue;**

**private final String stringValue;**

**private final List<String> stringList;**

**Que: What are Singleton class and How to implementation?**

**Ans:** A singleton class in Java is a class that allows only one instance of itself to be created and provides a global point of access to that instance. It is used when you want to ensure that there is only one instance of a class throughout the entire application, typically to control access to shared resources or manage global configurations.

Singleton classes are useful for managing shared resources, such as configurations, database connections, or logging systems.

**Implementation Steps for a Singleton Class:**

* **Private Constructor:** The class's constructor is made private to prevent other classes from creating instances directly.
* **Static Instance Variable**: Inside the class, you have a private static variable that holds the single instance of the class.
* **Static Method to Access Instance:** You provide a public static method, often named getInstance(), that returns the single instance of the class. This method creates the instance if it hasn't been created yet or returns the existing instance.

**Que: In Java can main method be overloaded or overrided ?**

**Ans:** In Java, the main method can be overloaded but not overridden.

**1. Overloading the main Method:**

* Overloading means defining multiple methods in the same class with the same name but different parameters.
* Java allows overloading the main method by creating another method named main with different parameter types.

***public class MainClass {***

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

***System.out.println("This is the standard main method.");***

***}***

***public static void main(int arg) {***

***System.out.println("Overloaded main method with int argument.");***

***}***

***public static void main(String arg) {***

***System.out.println("Overloaded main method with String argument.");***

***}***

***}***

**2. Overriding the main Method:**

* Overriding means providing a new implementation for a method inherited from a superclass.
* The main method in Java is static, which means it belongs to the class and is not overridden in subclasses.
* If a subclass defines a method with the same signature as the main method of the superclass, it's considered a separate method and does not override the main method.

**Que: What are Wrapper Class in Java?**

**Ans**: As we know java is not completely object oriented because it has primitive datatypes.

Wrapper classes in Java are used to convert primitive data types into objects (boxing) so that they can be included in collections, passed as arguments in method invocations that require objects, and used with other class-related functionalities. Each primitive data type in Java has a corresponding wrapper class. Here are the wrapper classes for primitive data types:

1. Byte: java.lang.Byte (Byte data type)
2. Short: java.lang.Short (Short data type)
3. Integer: java.lang.Integer (int data type)
4. Long: java.lang.Long (long data type)
5. Float: java.lang.Float (float data type)
6. Double: java.lang.Double (double data type)
7. Character: java.lang.Character (char data type)
8. Boolean: java.lang.Boolean (boolean data type)

* We create instances of Integer wrapper class using both the constructor and the valueOf() method.
* We parse a string to an integer using the parseInt() method of the Integer class.

**Que: What are Covarient Return Types?**

**Ans:** Covariant types enable a subclass to override a method from its superclass with a return type that is a subclass of the original return type.

It allows for more flexibility and specificity when defining method return types in subclasses.

**Que: What is Static Import?**

**Ans:** Static import is a feature introduced in Java 5 that allows you to import static members (fields and methods) of a class directly into another class, making it possible to use those static members without qualifying them with the class name.

Example:

***import static java.lang.Math.PI;***

***import static java.lang.System.out;***

***public class StaticImportExample {***

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

***double radius = 5.0;***

***double area = PI \* radius \* radius;***

***out.println("Area of circle: " + area);***

***}***

***}***

**Que: What are UniCode?**

**Ans:** Unicode is a character encoding standard used in computing to represent text and symbols from various writing systems and languages. It is designed to be a universal character set that encompasses characters from most of the world's writing systems, including Latin, Cyrillic, Greek, Arabic, Chinese, Japanese, Korean, and many others. Unicode assigns a unique code point (numeric value) to each character, allowing computers to represent and process text in a consistent and standardized manner across different platforms and programming languages.

**Que: Java Memory Managements?**

**Ans:** In Java, the memory management system is divided into different generations, each serving a specific purpose to optimize the performance and efficiency of garbage collection. The primary generations in Java's memory management system are the Young generation and the Old generation (also known as Tenured generation). Additionally, in certain JVM implementations, there is also a third generation called the Permanent generation (or Metaspace in modern JVMs). Let's discuss each of these generations:

**1. Young Generation:**

* **Purpose**: The Young generation is where newly created objects are initially allocated. It is designed for short-lived objects that are expected to have a short lifespan.
* **Structure**:

**Edben** **Space**: The main part of the Young generation where new objects are initially allocated.

Survivor Spaces (S0 and S1): Two survivor spaces where objects that survive garbage collection in Eden Space are moved.

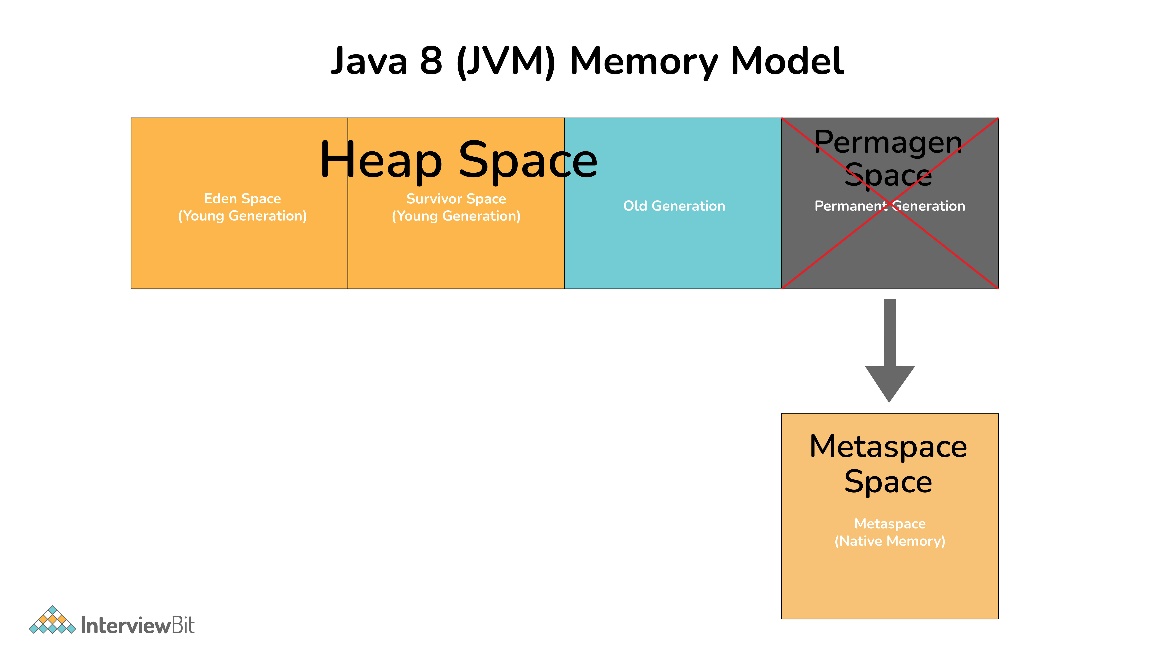
* **Objective**: The objective of the Young generation is to quickly identify and collect short-lived objects, minimizing the impact on the application's performance.

**2. Old Generation (Tenured Generation):**

* **Purpose**: The Old generation is where long-lived objects are stored. Objects that survive multiple garbage collection cycles in the Young generation are eventually promoted to the Old generation.
* **Structure**: The Old generation is a larger memory area compared to the Young generation, allowing it to store objects with longer lifespans without frequent garbage collection.
* **Objective**: The Old generation is optimized for managing long-lived objects efficiently and reducing the frequency of garbage collection cycles for these objects.

**3. Permanent Generation (Metaspace):**

* **Purpose**: In older versions of Java (up to Java 7), the Permanent generation (PermGen) was used to store metadata related to classes, methods, and other runtime information. Starting from Java 8, PermGen was replaced by Metaspace, which is a native memory area outside of the Java heap.
* **Structure**: Metaspace stores metadata about loaded classes, class structures, method information, and constants pool.
* **Objective**: Metaspace is responsible for managing class metadata efficiently, allowing for dynamic class loading and unloading without causing memory leaks or PermGen space errors.



**MetaSpace**: Java 8 stores the MetaData of classes in native memory called 'MetaSpace'. It is not a contiguous Heap Memory and hence can be grown dynamically which helps to overcome the size constraints. This improves the garbage collection, auto-tuning, and de-allocation of metadata.

**Que: What is a Memory Leak? Discuss some common causes of it.**

**Ans**: A memory leak in Java (or any programming language) occurs when a program allocates memory for objects or data structures but fails to release or deallocate that memory when it is no longer needed. As a result, the memory consumed by these unreleased objects keeps accumulating over time, leading to excessive memory usage, performance degradation, and eventually, system instability or crashes.