**Q-1: how to create an object in java**

soln:you need to define a class. A class is like a blueprint for objects. It defines what attributes (variables) and behaviors (methods) an object of that type will have.

Code:

public class MyClass {

// Attributes

private int myAttribute;

// Constructor

public MyClass(int value) {

this.myAttribute = value;

}

// Method

public void myMethod() {

System.out.println("Value of myAttribute: " + myAttribute);

}

}

Instantiate the Class : Once you have a class, you can create an object of that class using the new keyword followed by the class name and any necessary arguments to its constructor (if it has one).

**Q-2: what is the use of a new keyword in java**

Soln: the new keyword is used to create an instance of a class, which is essentially an object. When you use new, you're allocating memory for the object and initializing its attributes.

Here's a breakdown of how new works:

**Memory Allocation**: When you use new, Java allocates memory in the heap memory area for the object being created.

**Object Reference**: Finally, the new expression returns a reference to the newly created object. This reference can be assigned to a variable, passed as an argument, or used in any other context where an object reference is required.

Code:

public class MyClass {

private int myAttribute;

// Constructor

public MyClass(int value) {

this.myAttribute = value;

}

// Method

public void myMethod() {

System.out.println("Value of myAttribute: " + myAttribute);

}

}

public class Main {

public static void main(String[] args) {

// Creating an object of MyClass using new keyword

MyClass myObject = new MyClass(10);

// Using a method of the object

myObject.myMethod();

}

}

**Q-3: what are the different types of variables in java**

Soln:

**Local Variables:**

Local variables are declared within a method, constructor, or block.

They are accessible only within the scope they are declared in.

They must be initialized before they can be used.

Their lifespan is limited to the scope in which they are declared.

**Instance Variables (Non-Static Fields):**

Instance variables are declared within a class but outside any method, constructor, or block.

They are also known as non-static fields or member variables.

Each instance of the class (object) has its own copy of instance variables.

Instance variables are initialized with default values if not explicitly initialized.

They exist as long as the object they belong to exists.

**Static Variables (Class Variables):**

Static variables are declared with the static keyword within a class but outside any method, constructor, or block.

They are shared among all instances of the class.

Static variables belong to the class rather than any specific instance.

They are initialized once when the class is loaded and exist until the program terminates.

Static variables are accessed using the class name followed by the dot operator (e.g., ClassName.staticVariable).

**Q-4: what is the difference between instance variable and local variable**

Soln:

| **Instance Variable** | **Local Variable** |
| --- | --- |
| They are defined in class but outside the body of methods. | They are defined as a type of variable declared within programming blocks or subroutines. |
| These variables are created when an object is instantiated and are accessible to all constructors, methods, or blocks in class. | These variables are created when a block, method or constructor is started and the variable will be destroyed once it exits the block, method, or constructor. |
| These variables are destroyed when the object is destroyed. | These variables are destroyed when the constructor or method is exited. |
| It can be accessed throughout the class. | Its access is limited to the method in which it is declared. |
| They are used to reserving memory for data that the class needs and that too for the lifetime of the object. | They are used to decreasing dependencies between components I.e., the complexity of code is decreased. |

**Q-5: in which area memory is allocated for instance variable and local variable**

Soln: memory allocation for instance variables and local variables occurs in different areas:

**Instance Variables:**

Instance variables (also known as member variables) are declared within a class but outside any method, constructor, or block.

These variables are associated with an object of the class and have a separate copy for each instance (object).

Memory for instance variables is allocated in the heap.

The heap stores objects and their instance variables.

**Example**: If you have a class Person with an instance variable name, each Person object will have its own memory space for the name variable in the heap.

**Local Variables:**

Local variables are declared within a method, constructor, or block.

They are temporary and exist only within the scope of the method or block where they are defined.

Memory for local variables is allocated in the stack.

The stack stores method call frames, including local variables and method parameters.

**Example**: If you have a method calculateSum(int a, int b), the parameters a and b, as well as any other local variables within the method, are stored in the stack.

**Q-6: what is method overloading**

Soln: Method Overloading allows different methods to have the same name, but different signatures where the signature can differ by the number of input parameters or type of input parameters, or a mixture of both.

Method overloading in Java is also known as Compile-time Polymorphism, Static Polymorphism, or Early binding. In Method overloading compared to the parent argument, the child argument will get the highest priority.

Code:

public class Sum {

// Overloaded sum(). This sum takes two int parameters

public int sum(int x, int y) { return (x + y); }

// Overloaded sum(). This sum takes three int parameters

public int sum(int x, int y, int z)

{

return (x + y + z);

}

// Overloaded sum(). This sum takes two double

// parameters

public double sum(double x, double y)

{

return (x + y);

}

// Driver code

public static void main(String args[])

{

Sum s = new Sum();

System.out.println(s.sum(10, 20));

System.out.println(s.sum(10, 20, 30));

System.out.println(s.sum(10.5, 20.5));

}

}