**OOP FUNDAMENTAL ASSIGNMENT**

**MODULE - 12**

**1. How to create an object in java?**

To create an object in Java, you typically follow these steps:

Define a Class: First, you need to define a class. A class is a blueprint for creating objects. It specifies the attributes and behaviors of objects of that type.

Instantiate the Class: Once you have a class defined, you can create an object of that class using the new keyword followed by a call to the class constructor. The constructor initializes the object.

**2. What is the use of a new keyword in java?**

In Java, the new keyword is used to create new instances of classes. Its primary purpose is to dynamically allocate memory for an object of a class and initialize it by invoking a constructor.

Here's how the new keyword is typically used:

* Object Creation: It's used to create new objects of a class. When you write new ClassName(), you're instructing Java to allocate memory for an object of type ClassName and then call the constructor of ClassName to initialize the newly created object.
* Array Creation: The new keyword is also used to create arrays of objects or primitive data types.
* Anonymous Object Creation: Sometimes, you might create an object without assigning it to a reference variable. This is called creating an anonymous object.

**3. What are the different types of variables in java?**

* Local Variables: Variables declared within methods, constructors, or blocks. They are visible only within the scope they are declared in.
* Instance Variables (Non-Static Fields): Variables declared within a class but outside any method, constructor, or block. Each instance of the class has its own copy of instance variables.
* Static Variables (Class Variables): Variables declared with the static keyword within a class but outside any method, constructor, or block. They belong to the class rather than any specific instance of the class.

**4. What is the difference between instance variable and local variable?**

| **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. |
| These variables are given a default value if it is not assigned by code. | These variables do not always have some value, so there must be a value assigned by code. |
| It is not compulsory to initialize instance variables before use. | It is important to initialize local variables before use. |

**5. In which area memory is allocated for instance variable and local variable?**

In Java, memory for instance variables and local variables is allocated in different areas:

Instance Variables: Memory for instance variables is allocated on the heap. Each instance of a class has its own set of instance variables, and these variables are created when an object of the class is instantiated using the new keyword. Instance variables are part of the object's state and are accessible throughout the class.

Local Variables: Memory for local variables is allocated on the stack. Local variables are declared within methods, constructors, or blocks, and their scope is limited to the block in which they are declared. They are created when the block is entered and destroyed when the block is exited. Since the stack is a relatively smaller memory space and follows a last-in-first-out (LIFO) order, it's efficient for managing local variables with short lifetimes.

**6. What is method overloading?**

Method overloading in Java refers to the ability to define multiple methods in a class with the same name but with different parameter lists. This allows methods to perform similar tasks but with different types of input or different numbers of parameters.

When you overload a method, you keep the method name the same but vary the method's signature (i.e., the number, type, or order of parameters). Java determines which overloaded method to execute based on the number and types of arguments passed during the method invocation.