OOPs Fundamental Assignment

1.How to create an object in java

Sol-

In Java, to create an object:

Define a class that serves as a blueprint.

Use the new keyword to instantiate an object from the class.

Example

Class Name .object Name = new Class Name();

Class Name is the name of the class.

object Name is the name of the object.

The new keyword creates the object in memory and calls the class constructor to initialize it.

2. **what is the use of new keyword in java**

In Java, the new keyword is used to create new objects or allocate memory for new instances of a class. When you use new, Java will:

Allocate memory for the new object in the heap.

Call the constructor of the class to initialize the object.

Return a reference to the newly created object.

Common Uses of new:

Creating objects:

My Class obj = new My Class();

Here, new My Class() creates a new object of the My Class type, and the reference to this object is stored in the variable obj.

3.**what are the different type of variables in java**

Sol-1. Based on Data Type:

Primitive Data Types: These store simple values and are built into the language.

Byte: 8-bit integer

Short: 16-bit integer

Int: 32-bit integer

Long: 64-bit integer

Float: 32-bit floating-point number

Double: 64-bit floating-point number

Char: 16-bit Unicode character

Boolean: Represents true or false

4.**what is the difference between instance variable and local variables**

They are only accessible within the method or block in which they are declared.

1. **Scope**:

Instance Variable:

The scope of an instance variable is throughout the class. It can be accessed by any method or constructor of the class, using an instance of the class (object).

Local Variable:

The scope of a local variable is limited to the block or method where it is defined. Once the block or method execution ends, the local variable is destroyed.

2. **Lifetime**:

Instance Variable:

The lifetime of an instance variable lasts as long as the object it belongs to exists. If the object is created and destroyed, the instance variable is created and destroyed with it.

Local Variable:

The lifetime of a local variable lasts only during the execution of the method or block where it is declared. Once the method execution completes, the local variable is destroyed.

3. **Lifetime**:

Instance Variable:

The lifetime of an instance variable lasts as long as the object it belongs to exists. If the object is created and destroyed, the instance variable is created and destroyed with it.

Local Variable:

The lifetime of a local variable lasts only during the execution of the method or block where it is declared. Once the method execution completes, the local variable is destroyed.

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

Sol-

Instance Variables:

Memory Area: Heap Memory

Reason:

Instance variables are associated with objects. Every object has its own copy of instance variables.

When you create an object, the instance variables are stored in the heap, which is a region of memory reserved for objects.

The heap is where dynamic memory allocation takes place, and instance variables are allocated as part of the object on the heap.

Local Variables:

Memory Area: Stack Memory

Reason:

Local variables are temporary and are created when the method, constructor, or block is invoked.

They exist only for the duration of the method or block execution.

Stack memory is used to store method call frames, and local variables are stored in the stack. When the method or block completes, the memory is cleared.

Each time a method is called, a new stack frame is created for that method, and local variables are stored within that frame. Once the method returns, the frame (and the local variables) is destroyed.

6.**what is method overloading**

Method overloading in Java refers to the ability to define multiple methods with the same name but with different parameters (either different number of parameters or different types of parameters). It is a feature that allows a class to have more than one method with the same name but with different function signatures.

The main idea behind method overloading is to perform similar operations with different inputs using the same method name. The method signature must be different, which can be achieved by:

Changing the number of parameters.

Changing the type of parameters.

Changing the order of parameters (if the types are different).