Oops Fundamentals

Assignments Question-8

1. How to create an Object in Java ?

Ans: To create an object in Java, you first need to define a class that represents the blueprint for the object. Then you can create an instance of the class using the "new" operator followed by the class constructor.

Here's an example:

```
class ExampleClass {
  int value;

  ExampleClass(int v) {
    value = v;
  }
}

// Creating an object
ExampleClass obj = new ExampleClass(10);
```

n this example, we define a class "ExampleClass" with a constructor that takes an integer argument. We then create an instance of the class called "obj" and pass the value 10 to the constructor.

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

Ans: The "new" keyword in Java is used to create an instance of a class. The new operator dynamically allocates memory for an object and returns a reference to it. The reference is stored in a variable, which can then be used to access the object's properties and methods.

The general syntax to create an object using the new operator is:

```
ClassName objectName = new ClassName(arguments);
```

Where "ClassName" is the name of the class, "objectName" is the name of the reference variable that will refer to the created object, and "arguments" are the parameters passed to the constructor of the class.

For example:

```
class ExampleClass {
  int value;

  ExampleClass(int v) {
    value = v;
  }
}

// Creating an object
ExampleClass obj = new ExampleClass(10);
```

3.What are the difference between the types of variable in java ?

Ans: In Java, there are two types of variables: primitive variables and reference variables.

1. Primitive Variables: Primitive variables are the most basic data types in Java and include int, float, double, char, boolean, byte, short, and long. These variables are stored directly on the stack and hold their values directly.

2. Reference Variables: Reference variables, on the other hand, hold references to objects. They are stored on the heap and do not hold the object data directly. Instead, they hold the address of the object in memory.

Here are some key differences between primitive and reference variables in Java:

- Memory Allocation: Primitive variables are stored in the stack and take up a fixed amount of memory, while reference variables are stored in the heap and can grow dynamically as the program runs.

- Assignment: Primitive variables are assigned values directly, while reference variables are assigned references to objects, which can be changed during program execution.

- Equality: Primitive variables are compared by value, while reference variables are compared by reference. This means that two reference variables can point to the same object, but they are not equal unless they refer to the same reference.

- Nullability: Reference variables can be assigned the value "null", which means they do not reference any object, while primitive variables cannot be assigned the value "null".

In general, primitive variables are faster and more memory-efficient than reference variables, but they lack the flexibility and advanced features of reference variables.

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

Ans: Instance variables and local variables are two different types of variables in Java, and they have different scopes and lifetimes.

1. Instance Variables: Instance variables are defined in a class, outside of any method, and are associated with an instance of the class. Each object created from the class has its own separate set of instance variables, with their own values. Instance variables are also called fields or attributes.
2. Local Variables: Local variables, on the other hand, are defined within a method and are only accessible within the method in which they are declared. They are created when the method is called and are destroyed when the method returns. Local variables are stored in the stack, and their values can change with each method call.

Here are some key differences between instance and local variables in Java:

- Scope: Instance variables have a wider scope than local variables, as they are accessible from any method in the class, while local variables are only accessible within the method in which they are declared.
- Lifetime: Instance variables exist as long as the object exists, while local variables only exist for the duration of the method call.
- Initialization: Instance variables are automatically initialized to their default values (e.g. 0 for numeric types, null for reference types), while local variables must be explicitly initialized before they are used.
- Memory Allocation: Instance variables are stored in the heap and take up more memory than local variables, which are stored in the stack and take up less memory.

In general, it is a good practice to use local variables wherever possible, as they have a smaller memory footprint and a more limited scope, which makes them easier to debug and maintain. Instance variables should be used when you need to store data that is associated with an object and needs to be accessible from multiple methods.

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

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

1. Instance Variables: Instance variables are allocated on the heap, which is a dynamic memory area shared by all objects in a program. The heap is used to store objects and their associated data, and its size can grow dynamically as the program runs.
2. Local Variables: Local variables, on the other hand, are allocated on the stack, which is a static memory area used for storing method calls and local variables. The stack is used to store method arguments and local variables, and its size is determined at compile time. The

stack is used to store data that has a short lifespan and is discarded as soon as the method returns.

The allocation of memory on the stack and heap affects the performance and behavior of a program, as stack memory is faster and more limited in size compared to heap memory. In general, it is a good practice to use local variables whenever possible, as they have a smaller memory footprint and a more limited scope, which makes them easier to debug and maintain. Instance variables should be used when you need to store data that is associated with an object and needs to be accessible from multiple methods.

6.What is Methods Overloading ?

Ans: Method overloading in Java is a feature that allows a class to have multiple methods with the same name, as long as they have different parameters. This allows for greater code reusability and makes the code more readable and maintainable.

When a class has multiple methods with the same name, the Java compiler uses the number, types, and order of the parameters to determine which method to call at runtime. This process is called method resolution, and it is performed at compile-time.

Here are some key benefits of method overloading in Java:

- Code Reusability: Method overloading allows you to reuse the same method name for different purposes, making the code more readable and easier to maintain.
- Readability: By using the same method name for different purposes, the code becomes more readable and easier to understand.
- Type Checking: Method overloading allows the Java compiler to perform type checking at compile-time, ensuring that the correct method is called with the correct parameters.
- Flexibility: Method overloading allows you to provide multiple implementations of the same method, based on the type or number of parameters, making the code more flexible and adaptable.

In general, method overloading is a useful feature in Java, and it can be used to provide multiple implementations of the same method, based on the type or number of parameters. However, it is important to use method overloading judiciously, as having too many overloaded methods can make the code difficult to maintain and debug.