Assignment – 8th (Oops Fundamentals)

1. How to Create an Object in Java?

Ans: - In Java, objects are instances of classes and are created using the ***`new`*** keyword. Here's an example of how to create an object in Java:

```
class Student {
        String name;
        int rollNo;
        double avgGrade;

        public void displayInformation() {
                System.out.println("Name:" + name);
                System.out.println("Roll No:" + rollNo);
                System.out.println("Average:" + avgGrade);
        }
}

public class Main {
        public static void main(String [] args) {
                Student student = new Student();
                student.name = "Rahul Kumar";
                student.rollNo = 412;
                student.avgGrade = 80.0;

                student.displayInformation();
        }
}
```

Here, we have defined a class ***`Student`*** with instance variables ***`name`***, ***`rollNo`***, and ***`avgGrade`***, and a method ***`displayInformation()`***. In the main method, we create an instance of the ***`Student`*** class using the new operator and store it in the student variable. Then, we set the values for the instance variables of the student object. Finally, we call the ***`displayInformation()`*** method on the ***`student`*** object.

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

Ans: - The ***`new`*** keyword is used in Java to create an instance of a class. When you create an object using the ***`new`*** keyword, the Java Virtual Machine **(JVM)** allocates memory for the object on the heap, which is a part of memory that is used to store objects. The ***`new`*** keyword returns a reference to the newly created object, which you can store in a variable.

For example, you might have a ***`Car`*** class, with a constructor that takes the car's make and model as arguments. To create a new ***`Car`*** object, you would use the new keyword like this:

```
Car myCar = new Car("Toyota", "Camry");
```

3. What are the different type of variables in Java?

Ans: - In Java, there are three different types of variables: instance variable, class variable (static variable), and local variable.

1. **Instance variables: -** Instance variables are declared in a class, but outside of any method. They are associated with an instance of a class, and there is a separate copy of the variable for each object that is created from the class. Instance variables have default values, but you can also provide values when you create an object or later by using object reference variables.
2. **Class variables: -** Class variables are also declared in a class, but outside of any method, and are declared with the ***`static`*** keyword. There is only one copy of the class variable that is shared by all instances of the class. Class variables are often used to store values that are common to all objects of a class.
3. **Local variables: -** Local variables are declared within a method, constructor, or block of code. They are only accessible within the method, constructor, or block of code in which they are declared. Local variables do not have default values, so you must initialize them before you use them.

Here's an example to demonstrate the difference between the three types of variables:

```
class Student {
        // instance variable
        String name;

        // class variable
        static int totalStudents = 0;

        public Student(String name){
                this.name = name;
                totalStudents++;
        }

        Public void printName() {
                // local variable
                String message = “My name is ” + name;
                System.out.println(message);
        }
}

public class Main {
        public static void main(String [] args) {
                Student student1 = new Student(“Rahul Kumar”);
                Student student2 = new Student(“Aditya Kumar”);

                student1.printName();
                student2.printName();
                System.out.println(“Total students: ” + Student.totalStudents);
        }
}
```

Here, **`name`** *is an instance,* **`totalStudents`** *is a class(static), and* **`message'`** *is a local variable.*

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

Ans: - Instance variables are allocated memory in the heap, while local variables are allocated memory in the stack.

The heap is a region of memory where objects and their associated instance variables are stored. The heap is created when the Java Virtual Machine starts and is shared by all threads in the JVM. Because objects in the heap persist for the lifetime of the program, instance variables stored in the heap have a longer lifespan than local variables.

Local variables, on the other hand, are stored in the stack. The stack is created for each thread and stores method invocations and their associated local variables. The stack has a limited amount of memory, and local variables are automatically removed from the stack when the method invocation that created them completes. Because of the limited amount of memory in the stack, local variables have a shorter lifespan than instance variables.

5. What is method overloading?

Ans: - Method overloading in Java is a feature that allows a class to have multiple methods with the same name, but with different parameters. When a method is overloaded, the Java compiler determines which version of the method to call based on the number and types of arguments passed to the method. The idea behind method overloading is to increase the readability and reusability of the code by allowing methods to perform the same operations but with different parameters.

```
class Calculator {
        int add(int a, int b) {
                return a + b;
        }

        int add(int a, int b, int c) {
                return a + b +c;
        }

        double add(double a, double b) {
                return a + b;
        }
}
```

Here, the class ***`Calculator`*** has three methods with the same name add, but with different parameters. The first method takes two ***`int`*** parameters, the second takes three ***`int`*** parameters, and the third takes two ***`double`*** parameters. When you call the method ***`add`*** with different arguments, the Java compiler determines which method to call based on the number and types of arguments passed to the method.