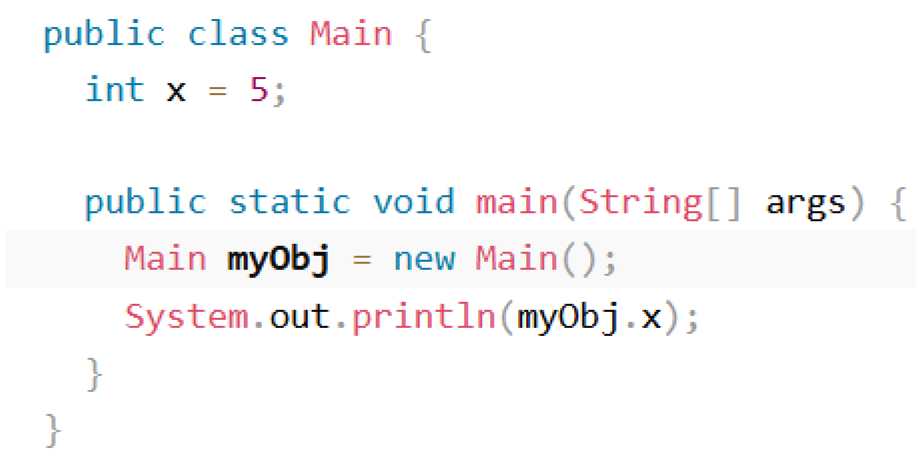
OOPS FUNDAMENTALS

Assignment Question

1. How to Create an Object in Java?
Answer → Firstly we have to create the class (or you can say the blueprint) then we have to create the instance of the class or it also called the object of the class.



```
public class Main {
  int x = 5;

  public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println(myObj.x);
  }
}
```

Note :-

1. Here myObj is called as Reference Variable it is not Actual Object of the class. It is only pointing to the actual Object of the class. Reference Variable use Stack Segment in Memory for its Memory allocation.
2. While new keyword is use to create the object of the class and Main() is the constructor of the class that is invoke while instance of the class is created. If user give it otherwise Compiler automatically invoke default constructor.
3. new Main() is the actual object of the class.

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

Answer → The new keyword is used to create instances (objects) of classes, allocating memory for the object and then calling the constructor to initialize it.

- **Object Creation:** The new keyword is fundamental to object-oriented programming in Java because it allows you to create objects from class blueprints.
- **Memory Allocation:** When you use new, Java allocates memory on the heap for the new object.
- **Constructor Invocation:** The new keyword automatically calls the constructor of the class to initialize the object's state.

Example :-

```
MyClass myObject = new MyClass();
```

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

Answer → In Java, variables are classified into three main types based on their scope and behaviour local variables, instance variables (also known as non-static variables), and static variables (also known as class variables).

- **Local Variables:**
  - Declared within a method, constructor, or block of code.
  - Accessible only within that specific method, constructor, or block.
  - Memory for local variables is released when the method or block finishes execution.
  - Local variables must be initialized before they are used.
- **Instance Variables (Non-Static Variables):**
  - Declared within a class but outside any method, constructor, or block.
  - Associated with an instance (object) of the class.
  - Each object of the class has its own copy of the instance variable.
  - Instance variables are created when an object of the class is instantiated and are accessible as long as the object exists.
- **Static Variables (Class Variables):**
  - Declared within a class using the static keyword.
  - Associated with the class itself, not with any specific object.
  - Shared among all instances (objects) of the class.
  - Static variables are initialized when the class is loaded and persist in memory throughout the program's runtime.

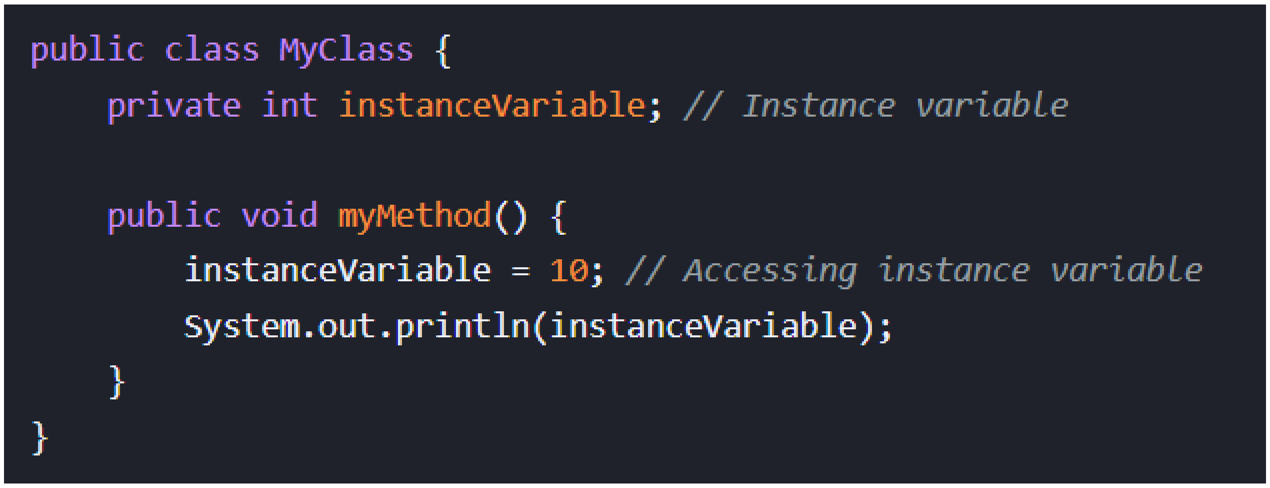
4. What is the difference between Instance variable and Local variables?

Answer → Instance variables belong to an object (instance) of a class and are accessible throughout the class, while local variables are declared within a method or block and are only accessible within that specific scope.

Instance Variables:

- **Scope:** Defined within a class but outside any method, constructor, or block.
- **Accessibility:** Accessible from any method, constructor, or block within the same class.
- **Lifetime:** Exist for the entire lifetime of the object (instance).

- **Memory:** Each object has its own copy of instance variables.
- **Example:-**

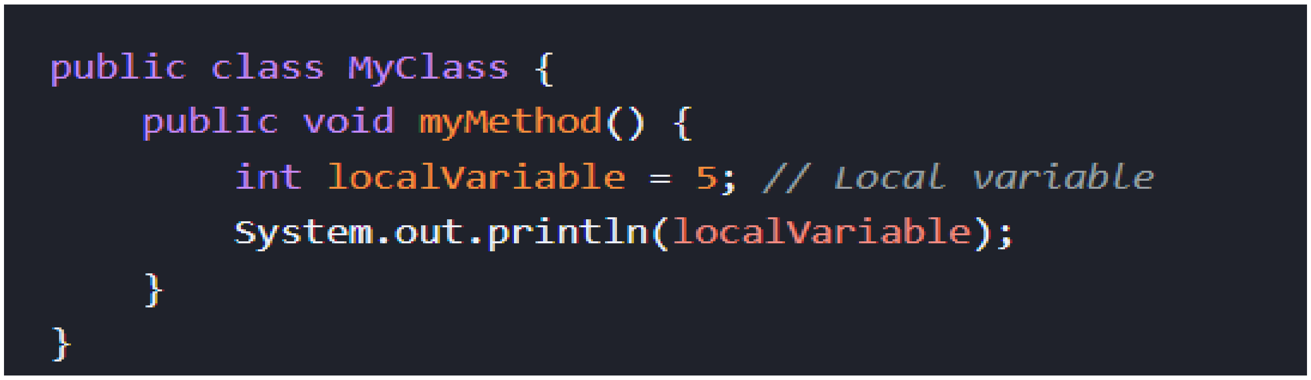


```
public class MyClass {
    private int instanceVariable; // Instance variable

    public void myMethod() {
        instanceVariable = 10; // Accessing instance variable
        System.out.println(instanceVariable);
    }
}
```

Local Variables:

- **Scope:** Declared within a method, constructor, or block.
- **Accessibility:** Only accessible within the method, constructor, or block where they are declared.
- **Lifetime:** Exist only during the execution of the method, constructor, or block.
- **Memory:** Local variables are created when the method is called and destroyed when the method finishes.
- **Example:**



```
public class MyClass {
    public void myMethod() {
        int localVariable = 5; // Local variable
        System.out.println(localVariable);
    }
}
```

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

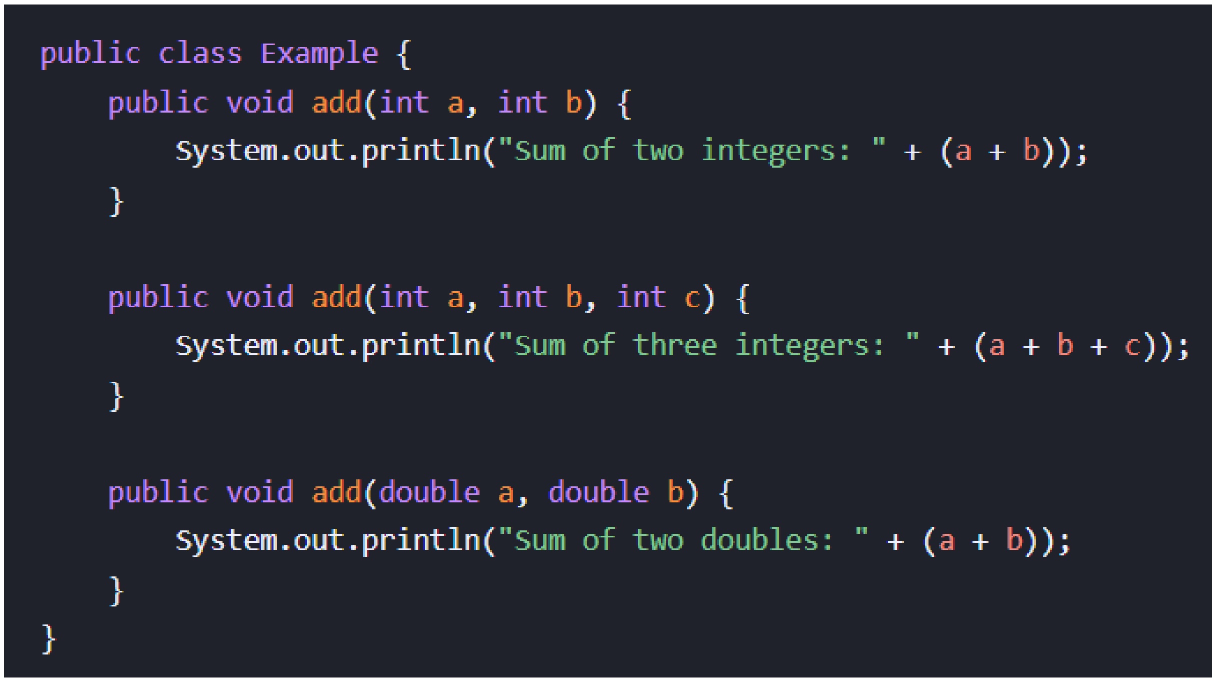
Answer → Instance variables in Java are stored in the heap memory, which is allocated to each individual object created from a class. Each object has its own copy of instance variables.

The stack segment is near the top of memory with high address. Every time a function is called, the machine allocates some stack memory for it. When a new local variables is declared, more stack memory is allocated for that function to store the variable. Such allocations make the stack grow downwards. After the function returns, the stack memory of this function is deallocated, which means all local variables become invalid. The allocation and deallocation for stack memory is automatically done. The variables allocated on the stack are called stack variables, or automatic variables.

6. What is method overloading ?

Answer → Method overloading allows you to define multiple methods within the same class that share the same name but have different parameters (number, type, or order). The compiler distinguishes these methods based on their parameter signatures, enabling you to call the appropriate method based on the arguments provided.

- **Same Name, Different Parameters:** The core concept of method overloading is having multiple methods with the same name, but each method accepts a unique set of parameters.
- **Compiler Resolution:** When you call a method, the compiler determines which overloaded method to execute based on the number, type, and order of the arguments you pass.
- **Example :-**



```
public class Example {
    public void add(int a, int b) {
        System.out.println("Sum of two integers: " + (a + b));
    }

    public void add(int a, int b, int c) {
        System.out.println("Sum of three integers: " + (a + b + c));
    }

    public void add(double a, double b) {
        System.out.println("Sum of two doubles: " + (a + b));
    }
}
```

Note :- the add method is overloaded. The compiler will choose the appropriate add method based on the number and type of arguments passed during the method call.