JAVA ASSIGNMENT 1

By – Satyam Verma

1) Explain the difference between primitive and reference data types with examples.

Answer –

Primitive Data Types

Primitive data types are used to store the basic values. They are referred as “Primitive” because they directly store the value into the memory, unlike the other storing methods like java objects or reference data types which refer to the location where the value is stored and we have to access the data from there.

Example -
int – Range -2147483648 to 2147483647

long – Range - -9223372036854775808 to 9223372036854775807

Usage –

```
public class PrimitiveExample {
    public static void main(String[] args) {
        int age = 25;        // Primitive type for integers
        double salary = 45000.99; // Primitive type for floating-point numbers
        boolean isEmployed = true; // Primitive type for booleans

        System.out.println("Age: " + age);
        System.out.println("Salary: " + salary);
        System.out.println("Is Employed: " + isEmployed);
    }
}
```

Reference Data Types

Reference data types store the references to the objects, thus the name “Reference”. They are often represented by complex objects like arrays, strings or custom objects. They don’t have a specific size; it depends on the user’s preference.

Usage -

```
public class ReferenceExample {
    public static void main(String[] args) {
        String name = "Alice"; // Reference type for strings
        int[] marks = {85, 90, 78}; // Reference type for arrays

        System.out.println("Name: " + name);
        System.out.print("Marks: ");
        for (int mark : marks) {
            System.out.print(mark + " ");
        }
    }
}
```

2) Explain the concept of encapsulation with a suitable example.

Answer –

Encapsulation is a fundamental OOP (Object Oriented Programming) principle that combines data (variables) and methods (functions) together into a single unit, a class. Encapsulation restricts the direct data access to the class variables instead, a provides controlled access to those data using methods usually getter and setter.

With Encapsulation we can ensure that –

1. The internal state of an object is protected from external modification.
2. Makes the code maintainable and flexible.
3. Access to the fields is controlled.

Encapsulation is implemented using Access modifiers in java. There are 4 access modifiers in java –

1. Default – The methods or data are accessible only within same package.
2. Private - The methods or data declared as private are accessible only within the class in which they are declared.
3. Protected - The methods or data declared as protected are accessible within the same package or subclasses in different packages.
4. Public - methods or data that are declared as public are accessible from everywhere in the program.

```
class Person {
    private String firstName;
    private String lastName;

    //Getter
    public String getName() {
        return firstName + " " + lastName;
    }

    //Setter
    public void setName(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }
}

public class OOP3 {
    public static void main(String[] args) {
        Person p = new Person();
        p.setName("Satyam", "Verma");
        System.out.println("Name - " + p.getName());
    }
}
```

3) Explain the concept of interfaces and abstract classes with examples.

Answer –

Interfaces Classes–

An Interface is a class used for making only “Abstract Class”, which can be then used by other classes to implement them. They are just empty bodies with data variables and methods declared but not defined. To implement the interface by classes “implements” keyword is used.

Example –

```
interface Animal {
    void makeSound();
    void sleep();
}
```

Implementation –

```
interface Animal {
    // Abstract method (no body)
    void makeSound();

    // Default method (with body)
    void sleep();
}

// Class implementing the interface
class Dog implements Animal {
    @Override
    public void makeSound() {
        System.out.println("Woof!");
    }

    @Override
    public void sleep() {
        System.out.println("Zzz");
    }
}
public class InterfaceClassExample {
  public static void main(String[] args) {
        Dog dog = new Dog();
        dog.makeSound();
        dog.sleep();
    }
}
```

Abstract Classes –

An abstract class is a restricted class which can’t be used to create objects instead they are used to make something like a blueprint or abstract of the class. These classes are used to make placeholder data variable and data methods they don’t need to be defined or have a body however it doesn’t mean they can’t have the body. These classes are extended by subclasses using the “extend” keyword.

```
abstract class Animal {
    // Abstract method (no body)
    public abstract void makeSound();

    // Concrete method (with body)
    public void sleep() {
        System.out.println("This animal is sleeping.");
    }
}

// Subclass of Animal
```

```java
class Dog extends Animal {
    // Override the abstract method
    @Override
    public void makeSound() {
        System.out.println("Woof!");
    }
}

public class AbstractClassExample {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.makeSound();
        dog.sleep();
    }
}
```