Assignment 17

Part A:

1. Functional Interface

A Functional Interface in Java is an interface that contains exactly one abstract method. It may contain multiple default and static methods, but only one abstract method is allowed. Functional interfaces were introduced in Java 8 to support Lambda Expressions.

Key Characteristics:

• Contains exactly one abstract method.

• Can have multiple default and static methods.

• Annotated with @FunctionalInterface (optional but recommended).

• Enables lambda expressions and method references.

Example:

```
@FunctionalInterface
interface MyFunctionalInterface {
   void show();
}

MyFunctionalInterface obj = () -> System.out.println("Hello");
obj.show();
```

Common built-in functional interfaces include Runnable, Callable, Predicate, Function, Consumer, and Supplier.

2. For-Each Loop

The for-each loop, also known as the enhanced for loop, is used to iterate over arrays and collections in a simplified manner without using index variables.

Syntax:

```
for (datatype variable : collection) {
   // code
}
```

Example:

```
int[] arr = {10, 20, 30};
```

```
for (int num : arr) {
   System.out.println(num);
}
```

Advantages:

• Cleaner and more readable code.

• No need for index management.

• Reduces risk of IndexOutOfBoundsException.

Limitations:

• Cannot access index directly.

• Cannot safely modify collection structure.

3. Difference Between break and continue

Both break and continue are control statements used in loops, but they behave differently.

break:

The break statement terminates the loop immediately and transfers control outside the loop.

```
for (int i = 1; i <= 5; i++) {
   if (i == 3)
      break;
   System.out.println(i);
}
```

continue:

The continue statement skips the current iteration and proceeds with the next iteration of the loop.

```
for (int i = 1; i <= 5; i++) {
   if (i == 3)
      continue;
   System.out.println(i);
}
```

4. if-else and switch Statements

if-else:

The if-else statement is used to execute a block of code based on a condition that evaluates to true or false.

```
int age = 18;

if (age >= 18) {
   System.out.println("Adult");
} else {
   System.out.println("Minor");
}
```

switch:

The switch statement is used when a variable needs to be compared against multiple fixed values.

```
int day = 2;

switch (day) {
   case 1:
      System.out.println("Monday");
      break;
   case 2:
      System.out.println("Tuesday");
      break;
   default:
      System.out.println("Invalid");
}
```

Difference Summary:

• if-else works with complex and range-based conditions.

• switch works with fixed value comparisons.

• switch improves readability for multiple exact matches.

5. Enhanced For Loop

The Enhanced For Loop is another name for the for-each loop introduced in Java 5. It is mainly used to traverse arrays and collections.

```
String[] names = {"Ram", "Shyam", "Mohan"};

for (String name : names) {
   System.out.println(name);
}
```

Advantages:

• Improves code readability.

• Reduces errors related to indexing.

• Ideal when index access is not required.

Part B:

1. Program to Demonstrate Method Overriding

```
class Animal {
   void sound() {
      System.out.println("Animal makes sound");
   }
}

class Dog extends Animal {
   @Override
   void sound() {
      System.out.println("Dog barks");
   }
}

public class MethodOverridingDemo {
   public static void main(String[] args) {
      Animal obj = new Dog();
      obj.sound();
   }
}
```

Output: Dog barks

2. Program to Demonstrate Method Overloading

```
class Calculator {

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

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

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

```
    }
}

public class MethodOverloadingDemo {
    public static void main(String[] args) {
        Calculator obj = new Calculator();

        System.out.println(obj.add(5, 3));
        System.out.println(obj.add(2.5, 3.5));
        System.out.println(obj.add(1, 2, 3));
    }
}
```

Output:

```
8
6.0
6
```

3. Program to Implement Abstract Class

```
abstract class Shape {

    abstract void draw();

    void display() {
        System.out.println("Displaying shape");
    }
}

class Circle extends Shape {

    void draw() {
        System.out.println("Drawing Circle");
    }
}

public class AbstractClassDemo {
    public static void main(String[] args) {
        Shape obj = new Circle();
        obj.draw();
        obj.display();
    }
}
```

Output:
Drawing Circle
Displaying shape

4. Program to Implement Interface

```
interface Animal {
   void sound();
}

class Cat implements Animal {

   public void sound() {
      System.out.println("Cat meows");
   }
}

public class InterfaceDemo {
   public static void main(String[] args) {
      Animal obj = new Cat();
      obj.sound();
   }
}
```

Output: Cat meows

5. Program to Implement Multiple Inheritance Using Interfaces

```
interface Father {
   void showFather();
}

interface Mother {
   void showMother();
}

class Child implements Father, Mother {

   public void showFather() {
      System.out.println("This is Father");
   }

   public void showMother() {
      System.out.println("This is Mother");
   }
}
```

```
public class MultipleInheritanceDemo {
   public static void main(String[] args) {
      Child obj = new Child();
      obj.showFather();
      obj.showMother();
   }
}
```

Output:
This is Father
This is Mother