LAB ACTIVITY 8:

ABSTRACT CLASS AND INTERFACE

Learning Outcomes

This Lab sheet encompasses 6 activities (Activity 8A, 8B, 8C, 8D,8E and 8F).

By the end of this tutorial session, you should be able to:

- **Implement abstract classes in Java program**
- **Classify the built-in interface class in Java Program**
- **Create Java program using following classes:**
    - **Implementing interface**

Activity 8A

Briefly explain abstract class in Java programs.

Abstract Class

Abstract class is a restricted class that cannot be used to create an object. If we want to access abstract class's code, it must be inherited from another class

State the rules to create abstract class and method

Rules of Abstract Class and Method

- Must be declared with an abstract keyword
- Cannot be instantiated
- It can have final method
- It can have abstract and non - abstract method
- It can have constructors and static method

Activity 8B

Activity Outcome: Illustrates the use of Abstract classes in java

Procedure:
Step 1: Type the following code
Step 2: Save the program
Step 3: Compile and execute the program
Step 4: Write the output

```
abstract class Arithmetic
{
      int a,b;
      abstract void calc();
}

class Add extends Arithmetic
{
      void calc()
      {
            System.out.println("The sum is " + (a+b));
      }
}

class Sub extends Arithmetic
{
      void calc()
      {
            System.out.println("The difference is " + (a-b));
      }
}



class Abst
{
      public static void main (String args[])
      {
            Add obj1 = new Add();
            obj1.a = 10;
            obj1.b = 6;
            obj1.calc();
            Sub obj2 = new Sub();
            obj2.a = 10;
            obj2.b = 6;
            obj2.calc();
      }
}
```

Code

```java
//Illustrate the use of Abstract classes in Java
abstract class Arithmetic {
    int a, b;

    abstract void calc();
}

class Add extends Arithmetic {
    void calc() {
        System.out.println("The sum is " + (a + b));
    }
}

class Sub extends Arithmetic {
    void calc() {
        System.out.println("The difference is " + (a - b));
    }
}

class Abst {
    Run | Debug
    public static void main(String[] args) {
        Add obj1 = new Add();
        obj1.a = 10;
        obj1.b = 6;
        obj1.calc();
        Sub obj2 = new Sub();
        obj2.a = 10;
        obj2.b = 6;
        obj2.calc();
    }
}
```

Output

```
The sum is 16
The difference is 4
```

Activity 8C

Activity Outcome: Illustrates the use of Abstract classes in java

Procedure :

Step 1: Key-in below program.

Bike12.java

```
class Bike12{
      abstract void run();
}
```

Step 2 : Observe the output.

Output:

```
Bike12.java:1: error: Bike12 is not abstract and does not override abstract method run() in Bike12
class Bike12 {
^
1 error
```

Step 3: How to correct the above error?

Correct error above.

Code

```
abstract class Bike12 {
    abstract void run();
}
```

ACTIVITY 8D

The following example illustrates the use of Abstract classes in java

Procedure :
Step 1: Type the following code
Step 2: Compile the program
Step 3: Identify the missing syntax and correct them
Step 4: Save the program
Step 5: Compile and execute the program
Step 6: Write the output

```
abstract class Shape           // abstract class
{
    abstract void draw();      // abstract method
}

class Triangle extends Shape  //sub class Rectangle
{
    void draw(){System.out.println("drawing triangle");}
    // Abstract method implemented in a subclass
}

class Circle1 extends Shape
{
}

class TestAbstraction1
{
   public static void main(String args[])
   {
       Shape s=new Triangle();
       s.draw();
   }
}
```

Code

```java
abstract class Shape { // Abstract class
    abstract void draw(); // Abstract method
}

class Triangle extends Shape { // sub class Rectangle
    void draw() {
        System.out.println(x: "Drawing Triangle");
        // Abstract method implemeted in a subclass
    }
}

class Circle1 extends Shape {
    void draw() {
        System.out.println(x: "Drawing Circle");
        // Abstract method implemeted in a subclass
    }
}

class TestAbstraction1 {
    Run | Debug
    public static void main(String[] args) {
        Shape s = new Triangle();
        Shape c = new Circle1();
        s.draw();
        c.draw();
    }
}
```

Output:

```
Drawing Triangle
Drawing Circle
```

Activity 8E

Activity Outcome: Demonstrates the implement of polymorphism and interface in Java Program.

Procedure:
Step 1: Type the following program.

```
interface Person
{
	public String getName();
}

class Student implements Person
{
	String name="Amirul";

	public String getName()
	{
		System.out.println("Student Name:" + name);
		return name;
	}
}

class Employee implements Person
{
	String name="Amirul";

	public String getName()
	{
		System.out.println("Employee Name:" + name);
		return name;
	}
}

public class Act9B
{
	public static void main( String[] args )
	{
		String temp;

		Student studentObject = new Student();
		Person ref= studentObject;
		Person ref2 = new Student();

		Employee employeeObject = new Employee();
		Person ref3 = employeeObject;
		Person ref4 = new Employee();

		System.out.println ("getName() method of Student class is called");
		temp= ref.getName();
		System.out.println( temp );
		//Person ref. points to an Employee object
		ref = employeeObject;
		System.out.println("getName() method of Employee class is called");
		temp = ref3.getName();
		System.out.println( temp );
	}
}
```

Step 2: Compile and execute the program

```java
interface Person {
    public String getName();
}

class Student implements Person {
    String name = "Amirul";

    public String getName() {
        System.out.println("Student Name:" + name);
        return name;
    }
}

class Employee implements Person {
    String name = "Amirul";

    public String getName() {
        System.out.println("Employee Name:" + name);
        return name;
    }
}

public class Act9B {
    Run | Debug
    public static void main(String[] args) {
        String temp;

        Student studentObject = new Student();
        Person ref = studentObject;
        Person ref2 = new Student();

        Employee employeeObject = new Employee();
        Person ref3 = employeeObject;
        Person ref4 = new Employee();

        System.out.println(x: "getName() method of Student class is called");
        temp = ref.getName();
        System.out.println(temp);
        // Person ref. points to an Employee object
        ref = employeeObject;
        System.out.println(x: "getName() method of Employee class is called");
        temp = ref3.getName();
        System.out.println(temp);
    }
}
```

Step 3: Write the output

```
getName() method of Student class is called
Student Name:Amirul
Amirul
getName() method of Employee class is called
Employee Name:Amirul
Amirul
```

Activity 8F

Activity Outcome Create Java program using implementing interface

Procedure:
Step 1: Type the following program.

```
// File name : Animal.java

interface Animal {

   public void eat ( );
   public void travel ( );
}
```

Step 2: Save the program as Animal.java
Step 3: Open new notepad. Type the following program.

```
// File name : MammalInt.java
class MammalInt implements Animal
{
   public void eat ( )
   {
      System.out.println ("Mammal eats");
   }

   public void travel ( )
   {
      System.out.println("Mammal travels");
   }

   public int noofLegs ( )
   {
       Return 0;
   }

   public static void main (String args[ ])
   {
      MammalInt m = new MammalInt ( );
      m.eat ( );
      m.travel ( );
   }

}
```

Step 2: Compile and execute the program

```
class MammalInt implements Animal {
    public void eat() {
        System.out.println(x: "Mammal eats");
    }

    public void travel() {
        System.out.println(x: "Mammal travels");
    }

    public int noofLegs() {
        return 0;
    }

    Run | Debug
    public static void main(String args[]) {
        MammalInt m = new MammalInt();
        m.eat();
        m.travel();
    }
}
```

Step 3: Write the output

```
Mammal eats
Mammal travels
```