 Contact: 8087883669

Abstract Classes:

1. An abstract class in Java is a class that cannot be instantiated, meaning you cannot create objects of an abstract class directly.
2. The main purpose of an abstract class is to provide a common interface and functionality to its subclasses.
3. Abstract classes can have both abstract and non-abstract methods.
4. Abstract methods are declared without an implementation and must be overridden in the subclasses.
5. To define an abstract class, you use the abstract keyword in the class declaration and abstract keyword to method declaration

Inheritance:

1. Inheritance is a fundamental concept in object-oriented programming that allows a class to inherit properties and methods from another class.
2. The class that is being inherited from is called the superclass or parent class, and the class that inherits is called the subclass or child class.
3. Inheritance is represented using the extends keyword in Java.
4. The subclass can access the public and protected members (methods and variables) of the superclass.
5. Inheritance supports the concept of code reuse, as the subclass inherits the fields and methods defined in the superclass, reducing code duplication.

Abstract Class and Inheritance:

1. An abstract class can be used as a superclass for inheritance, providing a partial or complete implementation of the superclass behavior.
2. When a subclass extends an abstract class, it must provide implementations for all the abstract methods defined in the superclass.
3. The subclass can also override non-abstract methods inherited from the abstract class to modify their behavior.
4. Abstract classes can be used to define a common interface and shared functionality for a group of related classes.
5. You can create objects of the subclass, but not of the abstract class itself.

Overall, abstract classes and inheritance are powerful features in Java that allow you to create hierarchies of related classes, promote code reuse, and provide a common interface for subclasses. Abstract classes provide a way to define partial or complete implementation, while inheritance enables the subclass to inherit and extend the behavior of the superclass.

Abstract class example of Shape:

```
package util.example1;

public abstract class Shape {
   String name;
```



```
    public Shape(String name) {
        this.name = name;
    }
    public abstract void draw();

    public void printShapeName(){
        System.out.println("Shape name is : "+name);
        draw();
    }
}
```

Child of Shape class:

```
package util.example1;

public class Square extends Shape{
    public Square(String name) {
        super(name);
    }

    @Override
    public void draw() {
        System.out.println("Drawing square");
    }

    public void drawSquare(){
        System.out.println("Drawing square");
    }

}

package util.example1;

public class Triangle extends Shape{
    public Triangle(String name) {
        super(name);
    }

    public void drawTriangle(){
        System.out.println("Drawing triangle");
    }

    @Override
    public void draw() {
        System.out.println("Drawing triangle");
    }
```

 Contact: 8087883669

```
}
```

Runner class:

```
package util.example1;

import java.util.Scanner;

public class ShapeExample {

   public static void main(String[] args) {
      Shape shape = null;


      Scanner sc = new Scanner(System.in);

      System.out.println("Which shape do you want to print");
      String shapeType = sc.next();

      if(shapeType.equals("Triangle")){
         shape = new Triangle("MyTriangle");
      }else if(shapeType.equals("Square")){
         shape = new Square("MySquare");
      }else {
         System.out.println("Its not val valid input !!");
      }
      //Here null pointer exeption will come if we give invalid shape
      //Need to handle this npe
      if(shape != null) {
         shape.printShapeName();
      }

   }
}
```

Interface example:

Road Construction interface:

```
package util.construction;

public interface RoadConstruction {
   String experience();
   float budget();
   void takeRoadSpecs(String spec);
   void setLocation(String location);
```

 Contact: 8087883669

```
   String getProgress();
}
```

Implementation of two child classes of RoadConstruction :

```
package util.construction;

public class AbcLtdRoadBuilder implements RoadConstruction{
   private String roadSpec;
   private String location;
   private String vender;
   private String name;

   public AbcLtdRoadBuilder(String name) {
      this.name = name;
   }

   @Override
   public String experience() {
      return "constructed 10 roads";
   }

   @Override
   public float budget() {
      return 20000000.00f;
   }

   @Override
   public void takeRoadSpecs(String spec) {
      this.roadSpec = spec;
   }

   @Override
   public void setLocation(String location) {
      this.location = location;
   }

   @Override
   public String getProgress() {
      //ask vender progress
      return "20% completed";
   }

   public void someMethod(){
      System.out.println("Something");
   }
```

 Contact: 8087883669

```
    @Override
    public String toString() {
        return "AbcLtdRoadBuilder{" +
                "roadSpec='" + roadSpec + '\'' +
                ", location='" + location + '\'' +
                ", vender='" + vender + '\'' +
                ", name='" + name + '\'' +
                '}';
    }
}
```

```
package util.construction;

public class DefLtdLoadBuilder implements RoadConstruction{
    private String roadSpec;
    private String location;
    private String vender;
    private String name;

    public DefLtdLoadBuilder(String name) {
        this.name = name;
    }

    @Override
    public String experience() {
        return "constructed 15 roads";
    }

    @Override
    public float budget() {
        return 10000000f;
    }

    @Override
    public void takeRoadSpecs(String spec) {
        this.roadSpec = spec;
    }

    @Override
    public void setLocation(String location) {
        this.location = location;
    }

    @Override
    public String getProgress() {
        return "completed 50%";
    }
```



```
    @Override
    public String toString() {
        return "DefLtdLoadBuilder{" +
                "roadSpec='" + roadSpec + '\'' +
                ", location='" + location + '\'' +
                ", vender='" + vender + '\'' +
                ", name='" + name + '\'' +
                '}';

    }
}
```

RoadConstruction interface runner class :

```
package util.construction;

public class RoadConstructionDemo {
    public static void main(String[] args) {
        //int sum = getSum(input1, input2);
        RoadConstruction selectedVender = releseTenderAndSelectContractor();

//        AbcLtdRoadBuilder builder = new AbcLtdRoadBuilder("test");
//        builder.someMethod();

        String experience = selectedVender.experience();
        System.out.println("Experience is "+ experience);

        float budget = selectedVender.budget();
        System.out.println("budget is "+budget);

        selectedVender.setLocation("location");
        selectedVender.takeRoadSpecs("length=10km, width = 5m");

        System.out.println(selectedVender.toString());

    }

    private static RoadConstruction releseTenderAndSelectContractor() {

        RoadConstruction vender1 = new DefLtdLoadBuilder("Def limited");
        RoadConstruction vender2 = new AbcLtdRoadBuilder("Abc limited");

        //This is anonymous class declaration with interface implementation
        RoadConstruction vender3 = new RoadConstruction() {
            @Override
            public String experience() {
                return null;
```

 Contact: 8087883669

```
        }

        @Override
        public float budget() {
            return 0;
        }

        @Override
        public void takeRoadSpecs(String spec) {

        }

        @Override
        public void setLocation(String location) {

        }

        @Override
        public String getProgress() {
            return null;
        }
      };

      return vender1;
    }
}
//   Experience is constructed 10 roads
//      budget is 2.0E7
//      AbcLtdRoadBuilder{roadSpec='length=10km, width = 5m', location='location', vender='null', name='Abc
limited'}


//   Experience is constructed 15 roads
//      budget is 1.0E7
//      DefLtdLoadBuilder{roadSpec='length=10km, width = 5m', location='location', vender='null', name='Abc
limited'}
```

Assignment question:

Create interface called DatabaesConnection which will have below methods
connect method which will take string connection detail argument
disconnect method which will not take anything
fireQuery method which will take query and return string result
checkConnectionStatus method which will show the connectin status
(hint for this method, create boolean variable called isConnectionAlive and toggle it in connect and disconnect method)

Create two child implementer of this interface, OracleDatabaesConnection, and SybaseDatabaseConnnection and implement those methods.