Assignment 1: Shape Hierarchy
Objective
Create a shape hierarchy that uses an abstract class and interfaces to model different types of
shapes, such as circles and rectangles. Implement methods to calculate the area and perimeter of
each shape.
Requirements
1. Interface Shape:
o Declare the following methods:
 double calculateArea(): Method to calculate the area of the shape.
 double calculatePerimeter(): Method to calculate the perimeter of the shape.
2. Abstract Class AbstractShape:
o Implements the Shape interface.
o Contains a common attribute color and a method getColor() to return the color of
the shape.
o Abstract methods calculateArea() and calculatePerimeter().
 3.Concrete Classes:
 Circle:
o Implements calculateArea() to return the area of the circle.
o Implements calculatePerimeter() to return the perimeter (circumference) of the

circle.
 Rectangle:
o Implements calculateArea() to return the area of the rectangle.
o Implements calculatePerimeter() to return the perimeter of the rectangle.
 4.Driver Class ShapeDemo:
 Create instances of Circle and Rectangle, and display their area and perimeter.

sol:

```
package myPack;

interface Shape{
	double calculateArea();
	double calculatePerimeter();
}

abstract class AbstractShape implements Shape{
	String color;

	public AbstractShape() {

	}
	public AbstractShape(String color) {
		this.color=color;
	}
	public String getColor() {
		return color;
	}
```

```
        public abstract double calculateArea();
        public abstract double calculatePerimeter();
}
class Circle extends AbstractShape{

        int radius;
                public Circle() {
                super();
        }

        public Circle(String color, int radius) {
                        super(color);
                        this.radius = radius;
                }


        public double calculateArea() {
                return 3.14*radius*radius;
        }
        public double calculatePerimeter() {
                return 2*3.14*radius;
        }
}
class Rectangle extends AbstractShape{

        int length;
        int breath;

        public Rectangle() {
                super();
        }

        public Rectangle(String color, int length, int breath) {
                super(color);
                this.length = length;
                this.breath = breath;
        }

        public double calculateArea() {
                return length+breath;
        }

        public double calculatePerimeter() {
                return 2*(length+breath);
        }

}
public class ShapeDemo {
        public static void main(String[] args) {
                Circle circle=new Circle("yellow",5);
                System.out.println("             Circle");
                System.out.println("            ========");
                System.out.println("Circle              :"+circle.getColor());
                System.out.println("Area of Circle
```

```
:"+circle.calculateArea());
                System.out.println("Perimeter of Circle
:"+circle.calculatePerimeter());
                System.out.println();
                Rectangle rect=new Rectangle("red",10,5);
                System.out.println("               Rectangle");
                System.out.println("              ===========");
                System.out.println("Rectangle              :"+rect.getColor());
                System.out.println("Area of Rectangle
:"+rect.calculateArea());
                System.out.println("Perimeter of Rectangle
:"+rect.calculatePerimeter());
        }
}
```

Assignment 2: Vehicle Management System
Objective
Create a vehicle management system that models different types of vehicles using interfaces and
abstract classes. Implement methods to calculate fuel efficiency and travel time.
Requirements
1. Interface Vehicle:
o Declare the following methods:
 double calculateFuelEfficiency(): Method to calculate the fuel efficiency of the vehicle.
 double calculateTravelTime(double distance): Method to calculate the travel time for a given distance.
2. Abstract Class AbstractVehicle:
o Implements the Vehicle interface.
o Contains common attributes like brand and model, and a method getDetails() to return vehicle details.
o Abstract methods calculateFuelEfficiency() and calculateTravelTime(double distance).
3. Concrete Classes:
 Car:
o Implements calculateFuelEfficiency() to return the fuel efficiency of the car.
o Implements calculateTravelTime(double distance) to return the travel time for a car.
 Truck:
o Implements calculateFuelEfficiency() to return the fuel efficiency of the truck.
o Implements calculateTravelTime(double distance) to return the travel time for a
truck.
4.Driver Class VehicleDemo:
 Create instances of Car and Truck, and display their fuel efficiency and travel time.

sol:

```
package myPack;
interface Vehicle{
         double calculateFuelEfficiency();
         double calculateTravelTime();
}
abstract class AbstractVehicle implements Vehicle{
        public String model;
        public String brand;

public AbstractVehicle(String model, String brand) {
                super();
                this.model = model;
                this.brand = brand;
        }

public void getDetails() {
        System.out.println("Model of the vehicle
:"+model);
        System.out.println("Brand the Vehicle
:"+brand);
        }
        public abstract double calculateFuelEfficiency();
        public abstract double calculateTravelTime();
}

class Car extends AbstractVehicle {
        double distance;
        double fuelconsumed;
        double speed;

        Car(String model, String brand, double distance, double fuelconsumed,
double speed) {
                super(model, brand);
                this.distance = distance;
                this.fuelconsumed = fuelconsumed;
                this.speed = speed;
                System.out.println("FuelEfficiency of the car                  :
" + calculateFuelEfficiency());
                System.out.println("Time to cover the distance by car in hours
:" + calculateTravelTime());

                getDetails();
        }

        public double calculateFuelEfficiency() {
                return distance / fuelconsumed;
        }

        public double calculateTravelTime() {
                return distance / speed;
        }
}

class Truck extends AbstractVehicle {
```

```
        double distance;
        double fuelconsumed;
        double speed;

        Truck(String model, String brand, double distance, double fuelconsumed,
double speed) {
                super(model, brand);
                this.distance = distance;
                this.fuelconsumed = fuelconsumed;
                this.speed = speed;
                System.out.println("FuelEfficiency of the truck
: " + calculateFuelEfficiency());
                System.out.println("Time to cover the distance by truck in hours
: " + calculateTravelTime());

                getDetails();
        }

        public double calculateFuelEfficiency() {
                return distance / fuelconsumed;
        }

        public double calculateTravelTime() {
                return distance / speed;
        }
}

class VehicleDemo {
        public static void main(String ar[]) {
                Vehicle car1 = new Car("i20", "Hyundai", 1000, 150, 80);
                System.out.println();

                Truck t1 = new Truck("benz", "Ashok", 8000, 500, 70);
        }
}
```

Assignment 3: Animal Sound System
Objective
Create an animal sound system that uses interfaces and abstract classes to model different types of
animals and their sounds.
Requirements
1. Interface Animal:
o Declare the following methods:
 void makeSound(): Method to make a sound specific to the animal.
 String getType(): Method to get the type of the animal (e.g., mammal, bird).
2. Abstract Class AbstractAnimal:
o Implements the Animal interface.
o Contains common attributes like name and age, and a method getDetails() to return
animal details.
o Abstract method makeSound().
3. Concrete Classes:

 Dog:
o Implements makeSound() to print the sound of a dog.
o Implements getType() to return "Mammal".
 Parrot:
o Implements makeSound() to print the sound of a parrot.
o Implements getType() to return "Bird".
4. Driver Class AnimalDemo:
 Create instances of Dog and Parrot, and display their sounds and types.
(Driver class means Main class)

sol:

```
package myPack;

interface Animal {
	void makeSound(String sound);

	String getType();
}

abstract class AbstractAnimal implements Animal {
	String name;
	int age;

	public AbstractAnimal(String name, int age) {
		this.name = name;
		this.age = age;
	}

	void getDetails() {
		System.out.println("Name of the animal     : " + name);
		System.out.println("Age of the animal      : " + age);
	}

	public abstract void makeSound(String sound);

	public abstract String getType(String type);
}

class Dog extends AbstractAnimal {

	String type;
	String name;
	int age;

	Dog(String name, int age, String type) {
		super(name, age);
		this.type = type;
		System.out.println("Type of the animal    : " + getType());
	}

	public void makeSound(String sound) {
		System.out.println("Dog sounds            : " + sound);
```

```
        }

        public String getType() {

                return type;
        }

        @Override
        public String getType(String type) {
                return null;
        }
}

class Parrot extends AbstractAnimal {
        String type;
        String name;
        int age;

        Parrot(String name, int age, String type) {
                super(name, age);
                this.type = type;
                System.out.println("Type of the animal     : " + getType());
        }

        public void makeSound(String sound) {
                System.out.println("Animal sounds          : " + sound);
        }

        @Override
        public String getType(String type) {
                return type;
        }
        public String getType() {
                return type;
        }
}

public class AnimalDemo {

        public static void main(String[] args) {
                Dog d1 = new Dog("Sqooby", 3, "Mammal");
                d1.getDetails();
                d1.makeSound("Bow Bow");

                System.out.println();

                Parrot p1 = new Parrot("rasagula", 2, "warm-blooded
vertebrates");
                p1.getDetails();
                p1.makeSound("raucous");

        }
}
```

Assignment 4: Employee Management System

Objective
Create an employee management system that models different types of employees using interfaces
and abstract classes. Implement methods to calculate salaries and display employee details.
Requirements
1. Interface Employee:
o Declare the following methods:
 double calculateSalary(): Method to calculate the salary of the employee.
 String getDepartment(): Method to get the department of the employee.
2. Abstract Class AbstractEmployee:
o Implements the Employee interface.
o Contains common attributes like name and id, and a method getDetails() to return
employee details.
o Abstract method calculateSalary().
3. Concrete Classes:
o FullTimeEmployee:
 Implements calculateSalary() to return the salary of a full-time employee.
 Implements getDepartment() to return the department.
o PartTimeEmployee:
 Implements calculateSalary() to return the salary of a part-time employee.
 Implements getDepartment() to return the department.
4. Driver Class EmployeeDemo:
o Create instances of FullTimeEmployee and PartTimeEmployee, and display their salaries and details.

sol:

```
package myPack;

interface Employee{
	double calculateSalary();
	String getDepartment();
}

abstract class AbstractEmployee implements Employee{
	String name;
	int id;

	public AbstractEmployee() {
		super();
	}

	public AbstractEmployee(String name, int id) {
		super();
		this.name = name;
		this.id = id;
	}


	public void getDetails() {
		System.out.println("name"+name);
```

```
                System.out.println("ID"+id);
        }
        public abstract double calculateSalary();
        public abstract String getDepartment();
}

class FullTimeEmployee extends AbstractEmployee{
        double salary;
        String department;

        public FullTimeEmployee() {
                super();
        }

        public FullTimeEmployee(String name, int id, double salary, String
department) {
                super(name, id);
                this.salary = salary;
                this.department = department;
        }


        public double calculateSalary() {
                return salary;
        }

        @Override
        public String getDepartment() {
                return department;
        }
}
class  PartTimeEmployee extends AbstractEmployee{
        double salary;
        String department;

        public  PartTimeEmployee() {
                super();
        }

        public  PartTimeEmployee(String name, int id, double salary, String
department) {
                super(name, id);
                this.salary = salary;
                this.department = department;
        }


        public double calculateSalary() {
                return salary;
        }

        @Override
        public String getDepartment() {
                return department;
```

```
        }
}
public class EmployeeDemo{
        public static void main(String [] args) {
                FullTimeEmployee fte=new
FullTimeEmployee("Mani",007,90000,"Java");
                System.out.println("    FullTimeEmployee");
                System.out.println("    ==================");
                System.out.println("Employee Name  :"+fte.name);
                System.out.println("Employee Id    :"+fte.id);
                System.out.println("Salary         :"+fte.calculateSalary());
                System.out.println("Department     :"+fte.getDepartment());
                System.out.println();
                PartTimeEmployee pte=new
PartTimeEmployee("Pavan",005,70000,"Java");
                System.out.println("   FullTimeEmployee");
                System.out.println("  ==================");
                System.out.println("Employee Name :"+pte.name);
                System.out.println("Employee Id   :"+pte.id);
                System.out.println("Salary        :"+pte.calculateSalary());
                System.out.println("Department    :"+pte.getDepartment());
        }
}
```

Assignment 5: Bank Account System
Objective
Create a bank account system that models different types of bank accounts using interfaces and
abstract classes. Implement methods to calculate interest and display account details.
Requirements
1. Interface BankAccount:
o Declare the following methods:
 double calculateInterest(): Method to calculate the interest for the bank account.
 String getAccountType(): Method to get the type of the bank account (e.g., savings, checking).
2. Abstract Class AbstractBankAccount:
o Implements the BankAccount interface.
o Contains common attributes like accountNumber and balance, and a method getDetails() to return account details.
o Abstract method calculateInterest().
3. Concrete Classes:
o SavingsAccount:
 Implements calculateInterest() to return the interest for a savings account.
 Implements getAccountType() to return "Savings".
o CheckingAccount:
 Implements calculateInterest() to return the interest for a checking account.
 Implements getAccountType() to return "Checking".
4. Driver Class BankAccountDemo:
o Create instances of SavingsAccount and CheckingAccount, and display their interest
and details

sol:

```
package myPack;
interface BankAccount{
         double calculateInterest();
         String getAccountType();
}
abstract class AbstractBankAccount implements BankAccount{
        double accountNumber;
        double balance;
        public AbstractBankAccount() {
                super();
        }
        public AbstractBankAccount(double accountNumber, double balance) {
                super();
                this.accountNumber = accountNumber;
                this.balance = balance;
        }
        public void getDetails() {
                System.out.println("Account Details :"+accountNumber);
                System.out.println("Account Balance :"+balance);
        }
         public abstract double calculateInterest();
         public abstract String getAccountType();
}
class SavingsAccount extends AbstractBankAccount{
        String accountType="Savings";
        int interest=9;
        double time;

        public SavingsAccount(double accountNumber, double balance, double time)
{
                super(accountNumber, balance);
                this.time=time;
        }


        public double calculateInterest() {
                return balance*interest*time;
        }

        public String getAccountType() {
                return accountType;
        }

}
class CheckingAccount extends AbstractBankAccount{
        String accountType="Checking";
        int interest=15;
        double time;

        public CheckingAccount(double accountNumber, double balance, double
```

```
time) {
                super(accountNumber, balance);
                this.time=time;
        }


        public double calculateInterest() {
                return balance*interest*time;
        }

        public String getAccountType() {
                return accountType;
        }

}
public class BankAccountDemo {
        public static void main(String[] args) {
                SavingsAccount sa=new SavingsAccount(47637,90000,1);
                System.out.println("Account Number :"+sa.accountNumber);
                System.out.println("Account type   :"+sa.getAccountType());
                System.out.println("Balance        :"+sa.balance);
                System.out.println("Interest Rate  :"+sa.calculateInterest());
                System.out.println();
                CheckingAccount ca=new CheckingAccount(87487,40000,3);
                System.out.println("Account Number :"+ca.accountNumber);
                System.out.println("Account type   :"+ca.getAccountType());
                System.out.println("Balance        :"+ca.balance);
                System.out.println("Interest Rate  :"+ca.calculateInterest());
        }

}
```