PARVATHAM RAM CHARAN

Problem1:

Problem 1: Create an abstract class Shape with an abstract method double area(). Then, create two subclasses, Circle and Rectangle, that extend Shape and provide implementations for the area method. Write a main method to create instances of Circle and Rectangle, and display their areas.

Driver.java:

```
package problem1;
public class Driver {
    public static void main(String[] args) {
                Circle c = new Circle(2.5);

                Rectangle r = new Rectangle(2.5,10);

                System.out.println("Area of circle : "+c.area());
                System.out.println("Area of Rectangle : " +r.area());
        }
}
```

Shape.java:

```
package problem1;

public abstract class Shape {
   public abstract double area();
}
```

Circle.java:

```
package problem1;
public class Circle extends Shape{
   private double radius;
```

```
        public Circle(double radius) {
                this.radius = radius;
        }
        @Override
        public double area() {
                return  (3.14*radius*radius);
        }

}
```

Rectangle.java

```
package problem1;
public class Rectangle extends Shape{
    private double length;
    private double breadth;
    public Rectangle(double length,double breadth) {
        this.length=length;
        this.breadth=breadth;
    }
    @Override
    public double area() {
        return (length*breadth);
    }

}
```

Output:

```
<terminated> Driver (6) [Java Application] C:\Users\upendar parvatham\.p
Area of circle : 19.625
Area of Rectangle : 25.0
```

Problem 2: Create an abstract class Animal with an abstract method void sound(). Then, create three subclasses, Dog, Cat, and Cow, each implementing the sound method with their respective sounds. Write a main method to create instances of Dog, Cat, and Cow, and invoke the sound method on each instance.

Driver.java:

```
package problem2;
public class Driver {

	public static void main(String[] args) {
		Dog d = new Dog();
		Cat c = new Cat();
		Cow cw = new Cow();
		d.sound();
		c.sound();
		cw.sound();

	}

}
```

Animal.java:

```
package problem2;
public  abstract class Animal {
   public abstract void sound();
}
```

Cat.java

```
package problem2;
```

```
public class Cat extends Animal {
        @Override
    public void sound() {
            System.out.println("cat meows....");
    }
}
```

Cow.java:

```
package problem2;

public class Cow extends Animal{
        @Override
     public void sound() {
             System.out.println("cows baaa...");
     }
}
```

Dog.java:

```
package problem2;

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

Output:

```
Problems  Javadoc  Declaration  Console
<terminated> Driver (7) [Java Application] C:\Users\upendar parvatham\.p2\pool\plugins\org
dog barks.....
cat meows....
cows baaa...
```

Problem3:

Create an abstract class Appliance with fields for brand and power consumption, and an abstract method void turnOn(). Create three subclasses, WashingMachine, Refrigerator, and Microwave, each providing their own implementation of the turnOn method. Write a main method to create instances of WashingMachine, Refrigerator, and Microwave, and invoke the turnOn method on each instance to display brand and power consumed.

Driver.java

```
package problem3;
public class Driver {

        public static void main(String[] args) {
                WashingMachine w = new WashingMachine("LG",1400);
                Refrigerator r = new Refrigerator("Samsung",2000);
                Microwave m = new Microwave("Panosonic",2500);
                w.turnOn();
                r.turnOn();
                m.turnOn();
                //or
    Appliance wm = new WashingMachine("LG",1400);
    Appliance fridge = new Refrigerator("Samsung",2000);
    Appliance micro = new Microwave("Panosonic",2500);

    wm.turnOn();
    fridge.turnOn();
    micro.turnOn();
```

```
        }

}
```

Microwave.java:

```
package problem3;

public class Microwave extends Appliance{

        public Microwave(String brand, int powerConsumption) {
                super(brand, powerConsumption);
        }

        @Override
        public void turnOn() {
    System.out.println("Microwave (" + brand + ") is now ON. Power consumed: " +
powerConsumption + "W");

        }

}
```

Refrigerator.java

```
package problem3;

public class Refrigerator extends Appliance{

        public Refrigerator(String brand, int powerConsumption) {
                super(brand, powerConsumption);
        }

        @Override
        public void turnOn() {
    System.out.println("Refrigerator (" + brand + ") is now ON. Power consumed: " +
powerConsumption + "W");
```

```
        }



}
```

WashingMachine.java

```
package problem3;
public class WashingMachine extends Appliance{

        public WashingMachine(String brand, int powerConsumption) {
                super(brand, powerConsumption);
        }

        @Override
        public void turnOn() {
    System.out.println("Washing Machine (" + brand + ") is now ON. Power consumed: " +
powerConsumption + "W");

        }



}
```

Appliance.java

```
package problem3;
public abstract class Appliance {
    String brand;
    int powerConsumption;
    public Appliance(String brand,int powerConsumption){
        this.brand=brand;
```

```
            this.powerConsumption=powerConsumption;
    }
    public abstract void turnOn();
}
```

Output:

```
<terminated> Driver (8) [Java Application] C:\Users\upendar parvatham\.p2\pool\plugins\org.eclip
Washing Machine (LG) is now ON. Power consumed: 1400W
Refrigerator (Samsung) is now ON. Power consumed: 2000W
Microwave (Panosonic) is now ON. Power consumed: 2500W
Washing Machine (LG) is now ON. Power consumed: 1400W
Refrigerator (Samsung) is now ON. Power consumed: 2000W
Microwave (Panosonic) is now ON. Power consumed: 2500W
```

problem4: Task: Create an interface Animal with methods makeSound() and eat().

Implement this interface in two classes Dog and Cat.

Driver.java

```
package problem4;
public class Driver {
    public static void main(String[] args) {
            Animal dog = new Dog();  //Dog d = new Dog(); d.makeSound() also correct
        Animal cat = new Cat();

        dog.makeSound();
        dog.eat();

        cat.makeSound();
        cat.eat();
          }
}
```

Cat.java

```
package problem4;
public class Cat implements Animal {

	@Override
	public void makeSound() {
		System.out.println("cat meos");

	}

	@Override
	public void eat() {
		System.out.println("cat eat fish");

	}
}
```

Dog.java

```
package problem4;

public class Dog implements Animal {

	@Override
	public void makeSound() {
		System.out.println("dog braks: bow");
	}

	@Override
	public void eat() {
		System.out.println("dog eat bones");
	}
}
```

Animal.java

```
package problem4;


public interface Animal {
	public void makeSound();
	public void eat();
}
```

Output:

```
Problems  Javadoc  Declaration  Console
<terminated> Driver (9) [Java Application] C:\Users\upendar parvatham\.p2\pool\plugins\org.eclipse.justj.op
dog braks: bow
dog eat bones
cat meos
cat eat fish
```

Problem 5: Create an interface Vehicle with a default method startEngine() that prints "Engine started". Implement this interface in the class Car and override the startEngine() method.

Driver.java:

```
package problem5;
public class Driver {

	public static void main(String[] args) {
		Vechile  v = new Car();
		v.startEngine();

	}

}
```

Vechile.java

```
package problem5;
public interface Vechile {
        default void startEngine() {
          System.out.println("engine started");
        }
}
```

Car.java

```
package problem5;
public class Car implements  Vechile{

          @Override
          public void startEngine() {
      System.out.println("Car engine started with a roar!");
          }



}
```

Ouput:

```
Problems  Javadoc  Declaration  Console
<terminated> Driver (10) [Java Application] C:\Users\upendar parvatham\.p2\pool\plugins\
Car engine started with a roar!
```

Problem 6: Interface Inheritance - Create an interface Shape with methods draw() and calculateArea(). Create another interface Colorful that extends Shape and adds a method fillColor(). Implement these interfaces in the class Circle.

Driver.java

```
package problem6;
public class Driver {

        public static void main(String[] args) {
          Circle c = new Circle(2.5);
          c.draw();
          c.fillCOlor("red");
          c.calculateArea();




        }

}
```

Shape.java:

```
package problem6;
public interface Shape {
    public void draw();
    public void calculateArea();
}
```

Colorful.java

```
package problem6;
public interface Colorful extends Shape{
     public void fillCOlor(String color);
}
```

Circle.java:

```
package problem6;


public class Circle implements Colorful{
    private double radius;
    public Circle(double radius) {
            this.radius=radius;
    }
        public void draw() {
                System.out.println("draws...");
        }
        public void calculateArea() {
                double area = 3.14*radius*radius;
                System.out.println("area : "+area);
        }
        @Override
        public void fillCOlor(String color) {
      System.out.println("Filling the circle with color: " + color);
        }


}
```

Output:

```
<terminated> Driver (11) [Java Application] C:\Users\upendar parvatham\.p2\pool
draws...
Filling the circle with color: red
area : 19.625
```

Problem 7: Interface with Multiple Implementations - Create an interface Payment with a method pay(). Implement this interface in two classes CreditCardPayment and PaypalPayment. Write a PaymentProcessor class that uses the Payment interface to process payments.

Driver.java

```
package problem7;
public class Driver {

	public static void main(String[] args) {
		Payment cc = new CreditCardPayment();
		Payment pp = new PaypalPayment();

		cc.pay();
		pp.pay();



	}

}
//or
//PaypalPayment p = new PaypalPayment();
//CreditCardPayment c = new CreditCardPayment();
//
//PaymentProcesssor processor1 = new PaymentProcesssor(p);
//processor1.processPayment(150);
//
//
//PaymentProcesssor processor2 = new PaymentProcesssor(c);
//processor2.processPayment(950);
```

//check git hub

Shape.java:

```
package problem7;
public interface Payment {
        public void pay();
}
```

CreditCardPayment.java

```
package problem7;
public class CreditCardPayment implements Payment{
      @Override
     public void pay() {
          System.out.println("payment wad done via CreditCardPayment");
      }
}
```

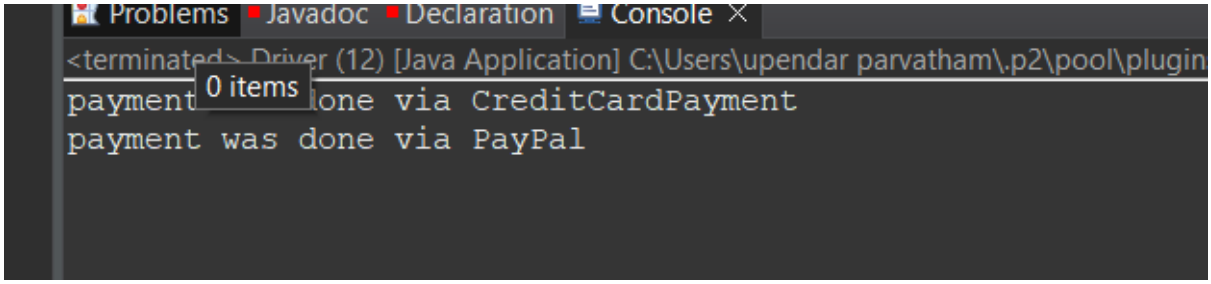
PaypalPayment.java:

```
package problem7;
public class PaypalPayment implements Payment{
         @Override
         public void pay() {
           System.out.println("payment was done via PayPal ");
         }
}
```

PaymentProcessor.java

```
package problem7;

public class PaymentProcessor {

	private Payment p;

	PaymentProcessor(Payment p){

		this.p = p;

	}

	public void pay() {

	  p.pay();

	}

}
```

Output:



Problem 8: Anonymous Inner Class Implementing an Interface - Create an interface Greeting with a method sayHello(). Write a method generateGreeting() in another class that uses an anonymous inner class to implement the Greeting interface and prints a greeting message

driver.java

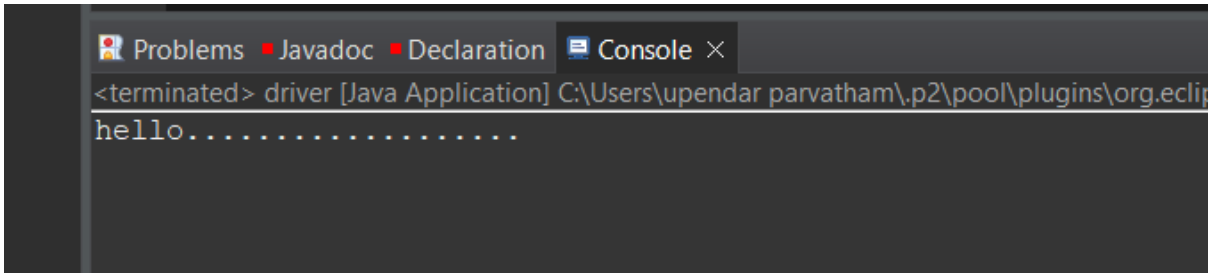
```
package problem8;

public class driver {

	public void genarateMeeting() {

		Greeting g  = new Greeting() {

			@Override
```

```
                public void sayHello() {
                        System.out.println("hello.................");
                }
        };
        g.sayHello();
    }
    public static void main(String[] args) {
            driver d = new driver();
            d.genarateMeeting();
        }
}
```

Greeting.java

```
package problem8;
public interface Greeting {
    public void sayHello();
}
```

Output:



Problem 9: Anonymous Inner Class Extending an Abstract Class - Create an abstract class Shape with an abstract method draw(). Write a method createShape() in another class that uses an anonymous inner class to extend Shape and provides an implementation for the draw() method.

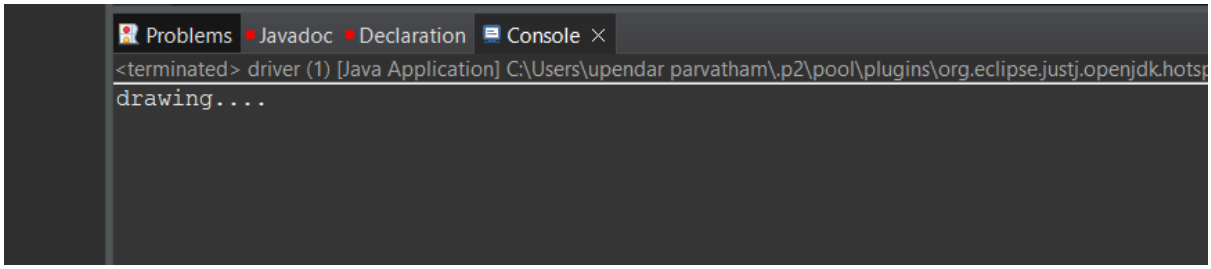
Driver.java

```
package problem9;
public class driver {
    public void createShape() {
        Shape s = new Shape() {
            public void draw() {
                System.out.println("drawing....");
            }
        };
        s.draw();
    }
    public static void main(String[] args) {
            driver d = new driver();
            d.createShape();
        }
}
```

Shape.java

```
package problem9;
public abstract class Shape {
    public abstract void draw();
}
```

Output:



Problem 10: Anonymous Inner Class Extending a Regular Class - Create a class Printer with a method printMessage(). Write a method createPrinter() in another class that uses an anonymous inner class to extend Printer and overrides the printMessage() method.

driver.java

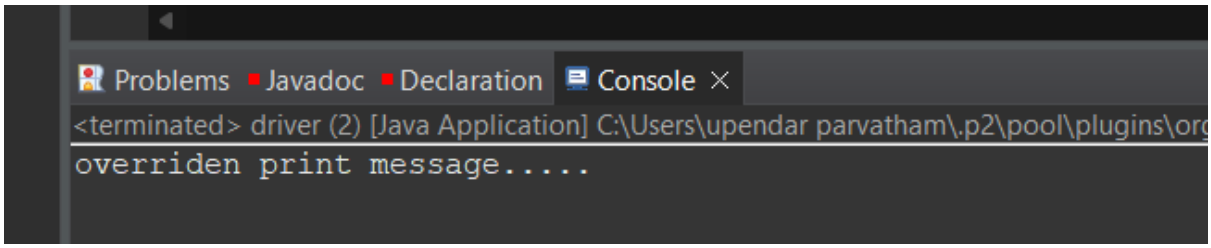
```
package problem10;
public class driver {
        public void createPrinter() {
             Printer p = new Printer() {
              @Override
              public void printMessage() {
                     System.out.println("overriden print message.....");
              }
             };
             p.printMessage();
        }
        public static void main(String[] args) {
                                driver d = new driver();
                                d.createPrinter();
                     }
}
```

Printer.java:

```
package problem10;
public class Printer {
   public void printMessage() {
          System.out.println("priting message.......");
    }
```

}

Output:


Problems
Javadoc
Declaration
Console
<terminated> driver (2) [Java Application] C:\Users\upendar parvatham\.p2\pool\plugins\org
overriden print message.....