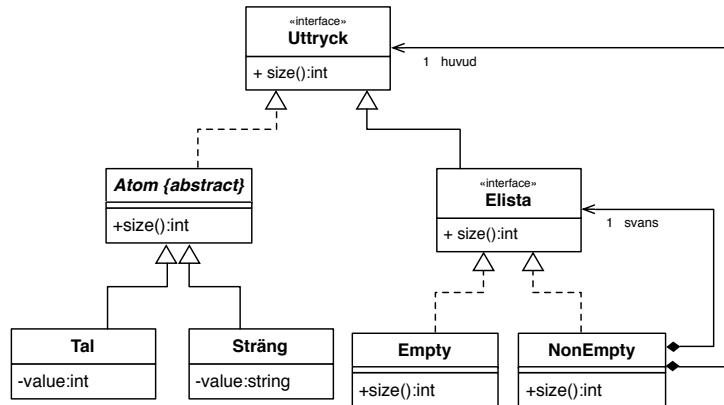


# Tentamen i Objektorienterad modellering och design

## Lösningar

### 1. a. Klassdiagram



### b. Null Object - mönstret

```

c. public interface Uttryck {
    public int size ();
}
public abstract class Atom implements Uttryck{
    public int size () {
        return 1;
    }
}
public interface Elista extends Uttryck{
}
public class Empty {
    public int size () {
        return 0;
    }
}
public class NonEmpty {
    Uttryck huvud;
    Elista svans;
    public int size () {
        return huvud.size () + svans.size ();
    }
}
  
```

```

2. a. public class Circle {
    private Point centre;
    private int radius;
    public void move(int dx, int dy) {
        centre.move(dx, dy);
    }
}
public class Point {
    private int x,y;
    public void move(int dx, int dy) {
        x += dx;
        y += dy;
    }
}
public class Drawing {
    private List<Circle> circles;
    //omissions
  
```

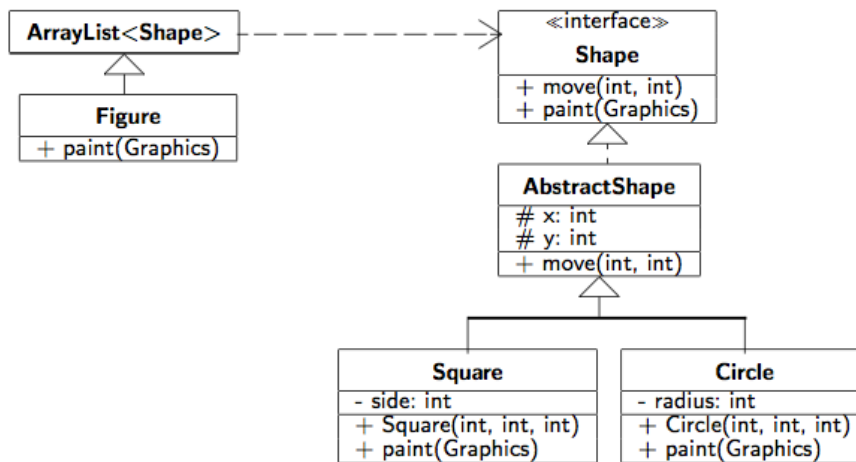
```

private void moveCircle(Circle aC, int dx, int dy) {
    aC.move(dx, dy);
}
}

```

- b. (a) Bryter inte mot *Lokalitetsprincipen*, dvs operationer implementeras där operanderna är lättast tillgängliga.
- (b) *Bättre integritet*. Klassen Point och klassen Circle behöver inte ge ut (privata) attribut.
- (c) Drawing *mindre beroende* på Circle och Point (Drawing inte alls beroende av Point)
- (d) Möjliggör att göra Point till ett gränssnitt och därmed kunna ha många olika implementeringar av Point, t ex Point2D och Point3D.

3. a. Klassdiagram



```

public interface Shape {
    public void move(int dx, int dy);
    public void paint(Graphics gr);
}

public abstract class SimpleShape implements Shape {
    private int x, y;
    public void move(int dx, int dy) {
        x += dx;
        y += dy;
    }
}

public class Square extends SimpleShape {
    private int side;
    public Square(int x, int y, int side) {
        super(x,y);
        this.side = side;
    }
    public void paint(Graphics gr) {
        gr.drawRectangle(x,y,side,side);
    }
}

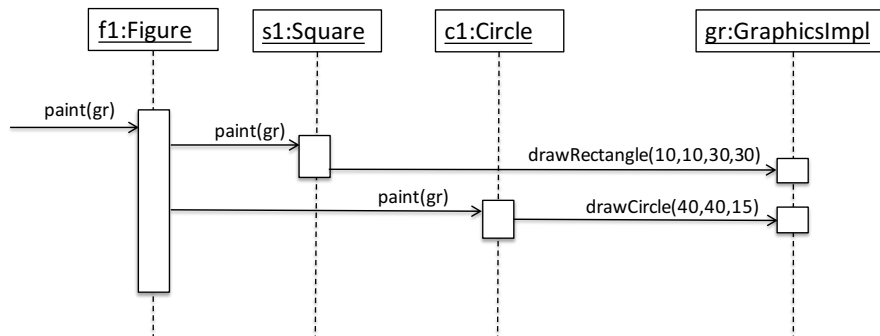
public class Figure extends ArrayList<Shape> {
    public void move(int dx, int dy) {
        for (Shape shape: this) {
            shape.move(dx, dy);
        }
    }
}

```

Circle är analog till Square.

```
b. public class Figure implements Shape {
    private List<Shape> list = new ArrayList<Shape>();
    public void move(int dx, int dy) {
        for (Shape shape: list) {
            shape.move(dx, dy);
        }
    }
    public void add(Shape newShape) {
        list.add(newShape);
    }
    public void paint(Graphics gr) {
        for (Shape shape: list) {
            shape.paint(gr);
        }
    }
}
```

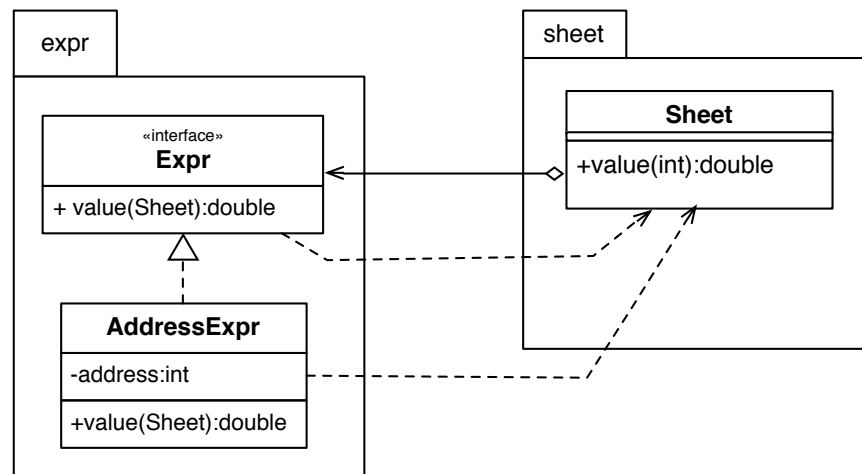
c. Sekvensdiagram



```
d. public class Drawing() {
    ShapeFactory sF = new ShapeFactory();
    //omissions
    Figure f1 = sF.create("Figure");
    Shape s1 = sF.create("Square 10 10 30");
    Shape c1 = sF.create("Circle 40 40 15");
    f1.add(s1);
    f1.add(c1);
    f1.paint(gr);
    //omissions
}

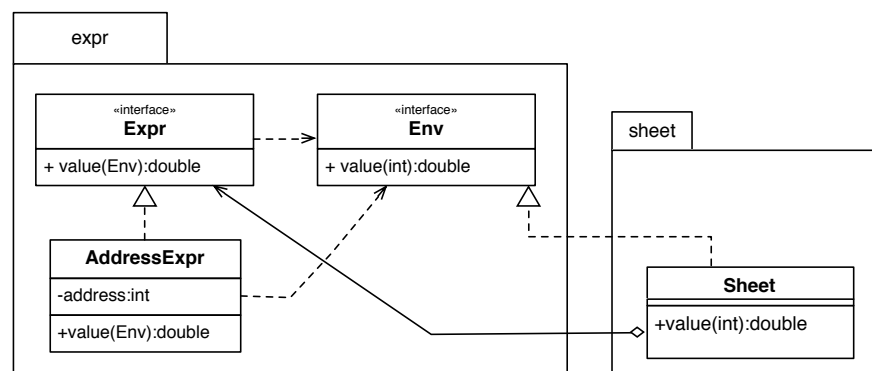
public class ShapeFactory {
    public Shape create(String str) {
        String[] tokens = str.split("\\s");
        if (tokens[0].equals("Figure"))
            return new Figure();
        else if (tokens[0].equals("Circle")) {
            int x = Integer.parseInt(tokens[1]);
            int y = Integer.parseInt(tokens[2]);
            int r = Integer.parseInt(tokens[3]);
            return new Circle(x, y, r);
        } else if (tokens[0].equals("Square")) {
            int x = Integer.parseInt(tokens[1]);
            int y = Integer.parseInt(tokens[2]);
            int s = Integer.parseInt(tokens[3]);
            return new Square(x, y, s);
        } else
            return null;
    }
}
```

4. a. UML-diagram



- b. (a) Alla klasser i cirkeln blir beroende av varandra. Alla klasser instabila.  
 (b) Kan inte ha separatkompilering. Om någon av klasserna ändras så måste alla kompileras om och testas på nytt.

c. UML-diagram



I paketet `expr`

```
public interface Expr {
    public double value(Env environment);
}
public interface Env {
    public double value(int address);
}
public class AddressExpr implements Expr {
    private int address;
    public double value(Env environment) {
        return environment.value(address)
    }
}
```

i paketet `sheet` finns

```
public class Sheet implements Env {
    private Expr[] array;
    public double value(int address) {
        return array[address].value(this)
    }
    // omissions
}
```