Klassediagrammer (II)

Modellering og Implementering

Michael R. Hansen

mrh@imm.dtu.dk

Informatics and Mathematical Modelling

Technical University of Denmark

Oversigt

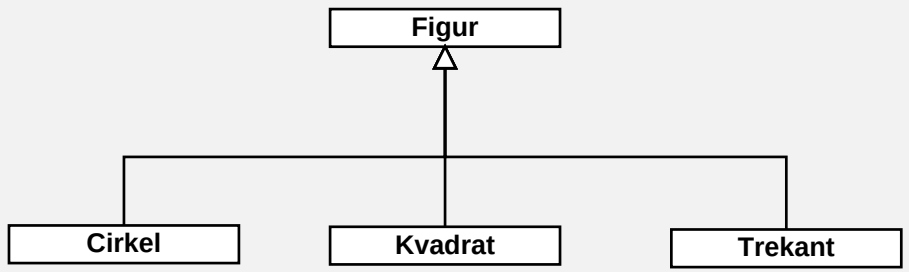
- nedarvning
- abstrakte klasser
- implementering, polymorfi

Begrebshierarki for simple Figurer (I)

En figur er enten en cirkel, en trekant eller et kvadrat.

Det generelle begreb er figur. De tre forskellige slags figurer: cirkel, trekant og kvadrat er specialiseringer.

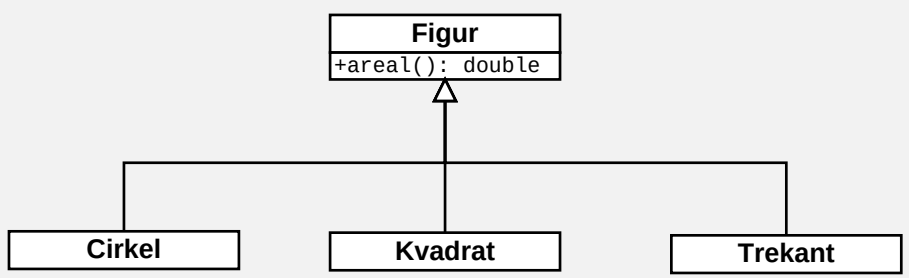
Dette *begrebshierarki*, med figur som det mest generelle begreb, beskrives i UML ved:



Begrebshierarki for simple Figurer (II)

Klassen Figur kaldes *superklasse* for klasserne Cirkel, Trekant og Kvadrat, som omvendt kaldes *subklasser* til Figur

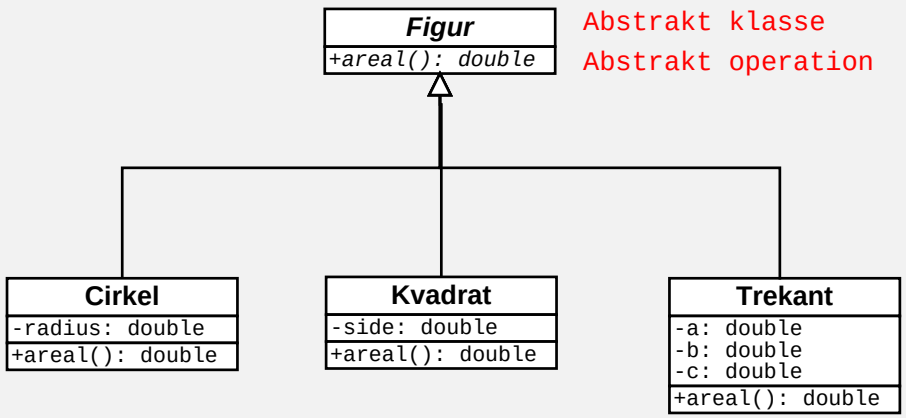
Subklasserne *nedarver* attributter og operationer fra superklassen. F.eks. cirkler, kvadrater og trekanter har alle en areal operation



Begrebshierarki for simple Figurer (III)

En figur er enten en cirkel, et kvadrat eller en trekant, så det er aldrig nødvendigt at oprette en instans af klassen Figur.

Denne klasse kan derfor laves *abstrakt*.



Implementering af figurklasserne

```
public abstract class Figur
{
  public abstract double areal();
}
```

```
public class Cirkel extends Figur
{ private double radius;

  public Cirkel(double r)
  { radius = r; }

  public double areal()
  { return Math.PI * radius * radius;
  }
}
```

```
public class Kvadrat extends Figur
{ private double side;

  public Kvadrat(double s)
  { side = s; }

  public double areal()
  { return side*side; }
}
```

```
public class Trekant extends Figur
{ private double a, b, c;
  public Trekant(double s1,..s2,..s3)
  { a = s1; b = s2; c = s3; }

  public double areal()
  { double s = (a+b+c)/2;
    return Math.sqrt(s*(s-a)
                     *(s-b)*(s-c)); }
}
```


Implementering: polymorfi

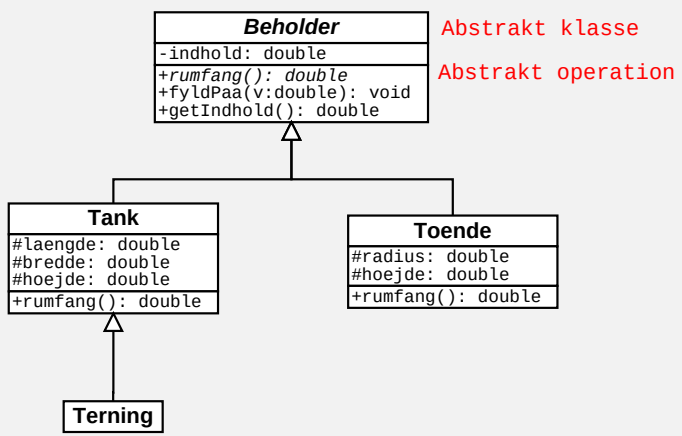
Klassehierarkiet giver mulighed for at konstruere *polymorfe* programmer der virker for alle slags figurer

```
public class Figurer
{  public static void main(String[] args)
   {  Figur[] figurer =
         new Figur[] { new Trekant(3,4,5), new Kvadrat(6),
                       new Cirkel(1), new Cirkel(3)
                     };
      for (int i=0; i<4; i++)
         System.out.println(figurer[i].areal());
   }
}
```

Udskriver

```
6.0
36.0
3.141592653589793
28.274333882308138
```


Begrebshierarki for simple Beholdere



fra *Java Precisely*, Peter Sestoft, MIT Press 2002

Implementering: Beholder

```
public abstract class Beholder
{
    private double indhold;

    public abstract double rumfang();

    public void fyldPaa(double v)
    {
        indhold = Math.min(rumfang(), indhold + v);
    }

    public double getIndhold()
    {
        return indhold;
    }
}
```


Implementering: Tank

Nedarver `fyldPaa` og `getIndhold`.

```
public class Tank extends Beholder
{   protected double laengde, bredde, hoejde;

    public Tank(double l, double b, double h)
    {
        laengde = l; bredde = b; hoejde = h;
    }

    public double rumfang()
    {
        return laengde * bredde * hoejde;
    }

    public String toString()
    {
        return "Tank (l,b,h) = (" + laengde + ", " + bredde + ", "+ hoejde + ")" ;
    }
}
```


Implementering: Terning

Nedarver `fyldPaa`, `getIndhold` og `rumfang`

```
public class Terning extends Tank
{
    public Terning(double side)
    {
        super(side, side, side);
    }

    public String toString()
    {
        return "Terning (s) = (" + laengde + ")" ;
    }
}
```

- superklassens konstruktor benyttes, idet en terning er en tank hvor $\mathtt{l} = \mathtt{b} = \mathtt{h}$

Implementering: Toende

Nedarver `fyldPaa` og `getIndhold`.

```
public class Toende extends Beholder
{
    protected double radius, hoejde;

    public Toende(double r, double h)
    {
        radius = r; hoejde = h;
    }

    public double rumfang()
    {
        return Math.PI * radius * radius * hoejde;
    }

    public String toString()
    {
        return return "Toende (r,h) = (" + radius + ", " + hoejde + ")" ;
    }
}
```

Opsummering

Klassehierarkier har såvel en rolle ved begrebsmodellering som ved programdesign.

- Felter og operationer skal indgå så højt oppe i hierarkiet som muligt. Dette forebygger gentagelse af kode.
- En abstrakt klasse kan implementere felter og operationer, der kan bruges i underklasserne.
- Abstrakte operationer implementeres i subklasserne.
- Understøtter polymorfi i programmer.