Repetition och abstraktion

Föreläsning 15

TDA540 - Objektorienterad Programmering

CHALMERS

Översikt

- Vad gör vi idag:
  - Repetition av objekt/klasser
  - Abstraktion
  - Lambda uttryck
- Laborationer: se till att allt är färdig v51!

Ett objekt . . .

... någonting man kan tänka på som en egen entitet

... någonting som gör något eller som man kan göra något med

... någonting som har egenskaper och/eller tillstånd

... någonting som representerar ett fysiskt föremål eller en verklig företeelse.

Exempel:

Om ni tittar er runt i föreläsningssalen ser in många olika fysiska föremål:

- studenter
- bord
- pennor
- böcker
- väskor
- lärare
- stolar
- datorer
- OH-apparater

. . .

Exempel: Bollar

En boll är ett föremål, som kan representeras som ett objekt i ett program.

Hur kan man beskriva en boll?

En boll beskrivs av sina *egenskaper*, såsom

- diameter
- färg
- elasticitet
- position
- . . .

Vi kan också beskriva en boll med hjälp dess *beteende*, dvs av vad man kan göra med bollen

- den kan kastas
- den kan studsas
- den kan rullas
- . . .

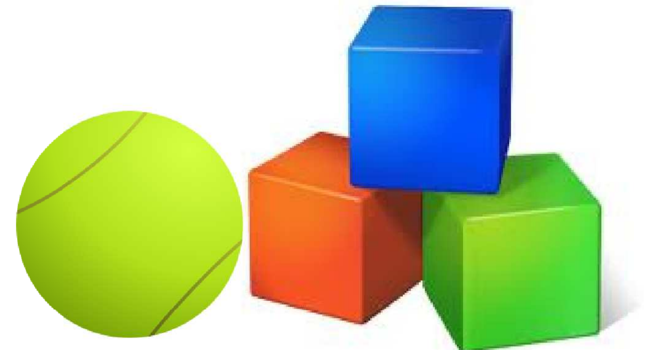
Exempel: Bollar

De *egenskaper* som beskriver ett objekt kallas för ***attribut*** och definierar objektets *tillstånd*. En specifik boll kan t.ex vara 24.2 cm i diameter, vara röd och ha en elasticitet på 74%. En annan boll kan vara 5.2 cm i diameter, vara vit och ha en elasticitet på 23%.

De aktiviteter man kan göra med ett objekt definierar objektets ***beteende***.

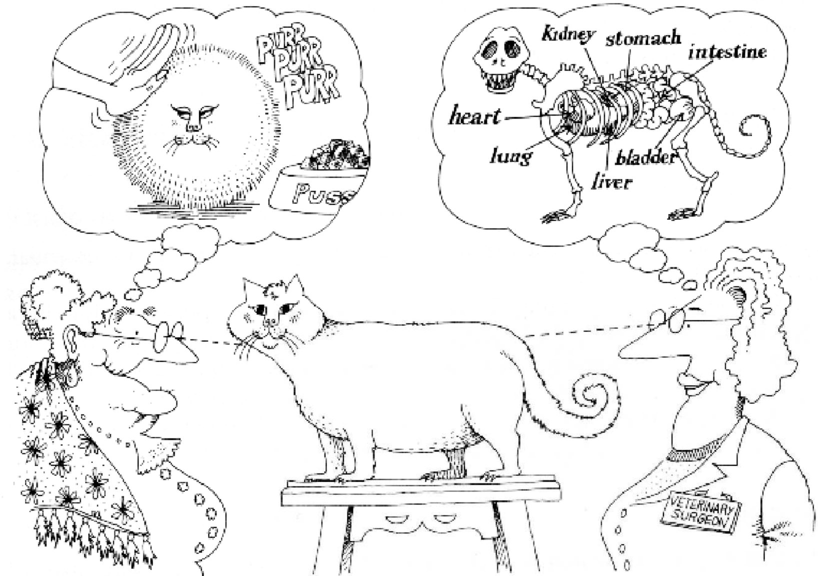
Alla objekt har en uppsättning attribut och en uppsättning beteenden.

Alla bollar har *samma uppsättning attribut och samma beteenden*. Det är dessa attribut och beteenden som skiljer bollar från andra objekt och som gör att "bollar är bollar"!



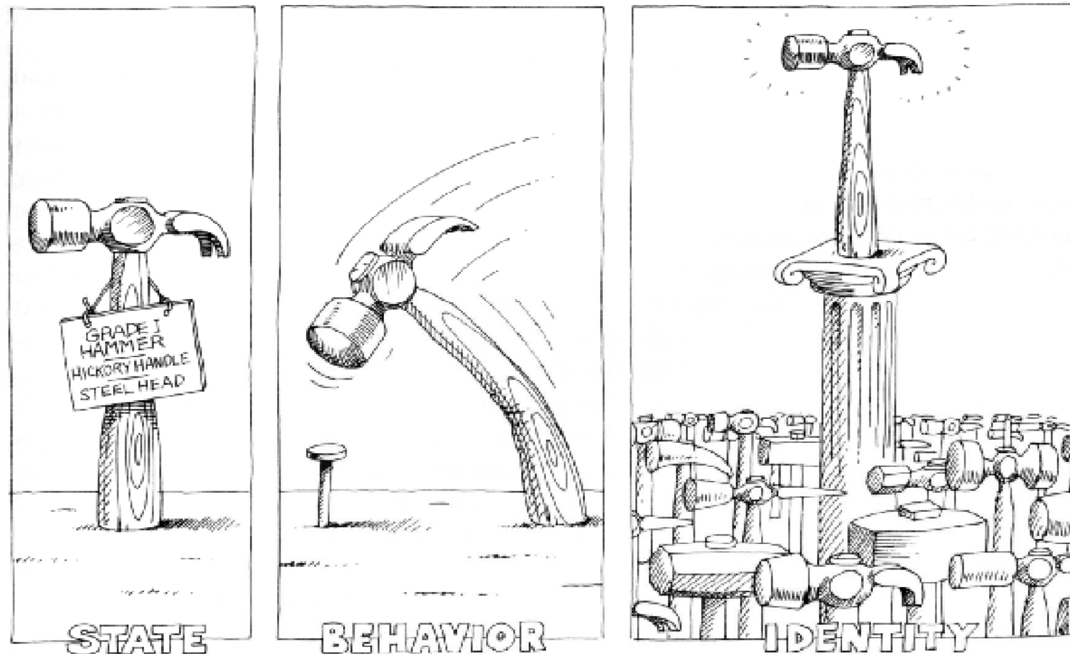
Objekt

Vilka attribut och beteenden hos en företeelse eller ett fysiskt föremålet som skall avbildas beror på vad man är intresserad av i den tänkta tillämpningen.



En modell är alltid en förenkling där man bortser från vissa omständigheter hos det som modelleras.

Objekt

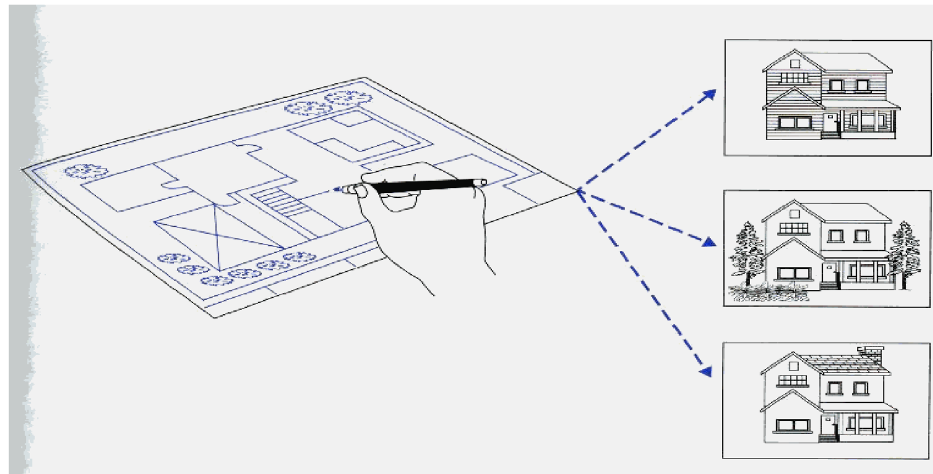


Ett objekt har ett *tillstånd*, ett *beteende* och en *identitet*, samt *tillhör en viss klass*.

Klasser

Vi konstaterade tidigare att alla bollar har *samma uppsättning attribut och samma beteenden*. Samt att det är dessa attribut och beteenden som gör att "bollar är bollar"! Man kan säga att en viss boll är ett objekt av klassen `Boll`.

En klass är en *mall* (*modell*, *mönster*, *ritning*, *beskrivning*) från vilken ett objekt kan skapas.



Klasser

Klasser är ett sätt för att skapa datatyper, för att kategorisera olika slag av data. En datatyp bestämmer vilka värden som tillhör datatypen och vilka operationer man kan utföra på dessa värden.

Att sammanföra data och de operationer som kan utföras på denna data till en enhet kallas för *inkapsling (encapsulation)*.

Idéen med en klass är att kunna modellera alla objekt av en viss typ på en och samma gång, istället för att modellera varje enskilt objekt var för sig.

Ett objekt är en *instans* av en klass.

I klassen beskrivs de attribut och de beteenden som alla objekt som tillhör denna klass skall ha.

Attribut beskrivs med hjälp av variabler. Beteenden beskrivs med hjälp av metoder.

Lite terminologi

De handlingar eller operationer som ett objekt av en viss klass kan utföra definieras således av de metoder som finns i klassen.

För att kunna använda ett objekt måste vi känna till dess *gränssnitt*, dvs vilka *tjänster* (vilken *service*) som objektet erbjuder och hur man får tillgång till dessa.

De tjänster ett objekt erbjuder bestäms av de metoder som finns definierade i klass som objektet tillhör.

När en metod anropas är det alltid två objekt involverade:

- *klient*, objektet som *anropar* metoden för att få service
- *server*, objektet som *exekverar* metoden för att leverera efterfrågad service.

Lite terminologi

Metoder är den primära mekanismen via vilken objekt kan *kommunicera* med och påverka varandra.

När en metod anropas, *skickar* klientobjektet ett *meddelande* med information till serverobjektet. Klientobjektet är alltså *avsändare* av meddelandet och serverobjektet är *mottagare* av meddelandet.

När metoden exekveras, utför serverobjektet de nödvändiga beräkningarna genom att använda sina egna attribut och/eller den information som skickas från klienten.

När en metod *terminerar* (exekverat klart) skickas information från servern till klienten. Klienten får alltså ett *svar* på sitt meddelande.

Lite terminologi

I en klass finns följande olika slag av attribut:

- *instansvariabler*, varje instans har sin egen unika uppsättning som beskriver instansens tillstånd
- *klassvariabler*, finns i endast en uppsättning som delas mellan samtliga instanser av klassen
- *konstanter* (både instanskonstanter och klasskonstanter kan finnas, men endast klasskonstanter skall användas).

I en klass finns två olika slag av metoder

- *instansmetoder*, varje instans har sin egen unika uppsättning
- *klassmetoder*, finns i endast en uppsättning som delas mellan samtliga instanser av klassen.

Klasser och objekt vi redan sett

Hittills i kursen har vi bland annat sett följande konstruktioner:

```
System.out.println(”Hello world! ”);
JOptionPane.showMessageDialog(null,"Hello world! ");
double tal = Double.parseDouble("10.123");
double tal2 = Math.pow(5, 2);
String str = "1 2 3 4 5 6";
Scanner scan = new Scanner(str);
int heltal = scan.nextInt();
double reelltTal = scan.nextDouble();
Robot robot = new Robot();
robot.move();
```

Ange samtliga klasser, objektreferenser, klassmetoder och instansmetoder i kodavsnittet ovan!

Klasser:

- System
- JOptionPane
- Double
- Math
- String
- Scanner
- Robot

Objektreferenser:

- out
- str
- scan
- robot

Klassmetoder:

- showMessageDialog
- parseDouble
- pow

Instansmetoder:

- println
- nextInt
- nextDouble
- move

Att skapa objekt

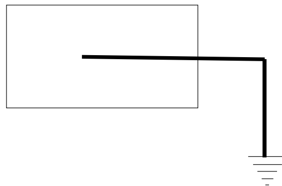
Objekt kan skapas från existerande klasser.

Detta görs genom att man först deklarerar en variabel av den klass man vill skapa ett objekt av:

KlassNamn objektNamn;

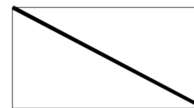
Deklarationen innebär att variabeln objektNamn är av klassen KlassNamn och får värdet **null** (då inget objekt ännu har skapats). Detta kan illustreras på följande sätt:

objectNamn



eller

objectNamn

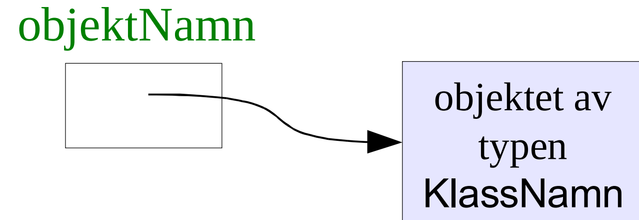


Att skapa objekt

För att faktiskt skapa ett objekt används (vanligtvis) operatorn **new** samt ett anrop av klassens *konstruktor*:

objektNamn = **new** KlassNamn();

vilket illustreras av bredvidstående figur:



Variabler som refererar till ett objekt, i likhet med variabeln objektNamn kallas för ***referensvariabler*** (medan variabler som används för att lagra enkla data typer kallas för ***enkla variabler***).

En konstruktor är en speciell metod för att initiera tillståndet hos det skapade objektet.

En konstruktor har samma namn som klassen själv.

En klass kan ha *flera konstruktorer*, med *olika* ***parameterprofiler***. Detta kallas för ***överlagring*** (*overloding*).

Det kan också finnas överlagrade metoder, dvs metoder med samma namn men med olika parameterprofiler.

Att definiera egna klasser

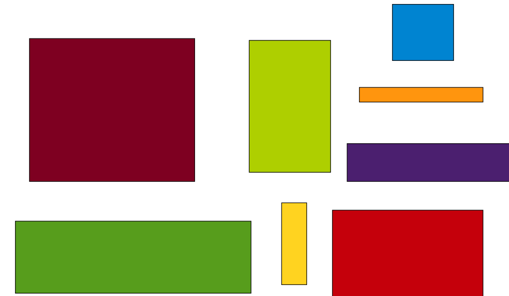
En deklaration av en klass kan innehålla deklarationer av

- *instansvariabler* (*tillståndsvariabler*), dvs de variabler som varje objekt av klassen har sin egen kopia av
- *konstruktorer*, dvs de mekanismer som används för att skapa ett nytt objekt ur klassen (och initiera dess tillstånd)
- *publika instansmetoder*, dvs de metoder som varje objekt ur klassen tillhandahåller för omvärlden
- *privata instansmetoder*, dvs lokala hjälpmetoder som bara används av andra metoder i klassen
- *klassvariabler* (*statiska variabler*), dvs de variabler som det endast finns en gemensam kopia av för alla objekt i klassen (kan vara publika eller privata)
- *klassmetoder* (*statiska metoder*), dvs metoder som inte är knutna till ett visst objekt utan till själva klassen (kan vara publika eller privata)
- *konstanter*, skall alltid göras statiska (kan vara publika eller privata).

Exempel: En klass för rektanglar

I ett program vill vi hantera rektanglar av olika utseende. De attribut vi är intresserade av hos en rektangel i vår tillämpning är:

- höjd
- bredd



Detta betyder att vi t.ex. bortser från vilken färg en rektangel har.

Det vi vill kunna göra med en rektangel, dvs de beteenden vi är intresserade av, är:

- ta reda på höjden
- ta reda på bredden
- ta reda på ytan
- ta reda på omkretsen
- ändra höjden
- ändra bredden
- skriva ut rektangelns tillstånd, dvs rektangelns höjd och bredd

När vi skapar en rektangel vill vi kunna välja att antingen få en ’standardrektangel’ eller få en rektangel för vilken vi anger tillståndet (dvs dess höjd och bredd). Det behövs alltså två olika konstruktorer.

Exempel: En klass för rektanglar

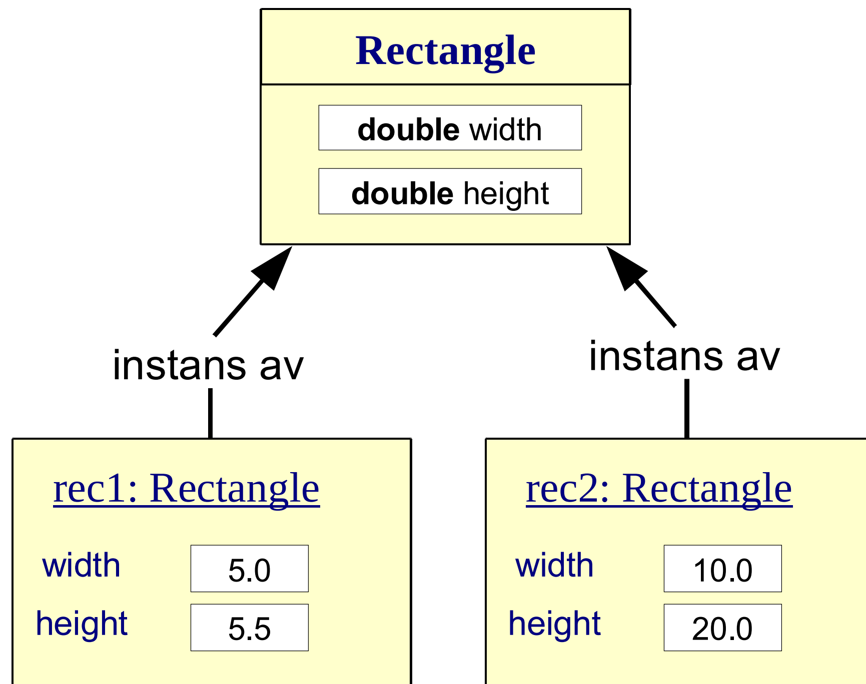
```
public class Rectangle {
  private double width;     //instansvariabel
  private double height;    //instansvariabel
  public Rectangle() {
    width = 0;
    height = 0;
  } //constructor
  public Rectangle(double w, double h) {
    width = w;
    height = h;
  }//constructor
  public double getWidth() {
    return width;
  }//getWidth
  public double getHeight() {
    return height;
  }//getHeight
  public void setWidth(double w) {
    width = w;
  } //setWidth
  public void setHeight(double h) {
    height = h;
  } //setHeight
  public double getArea() {
    return height * width;
  }//getArea
  public double getPerimeter() {
   return 2 * (height + width);
  } //getPerimeter
  public String toString() {
   return "Width = " + width + "\nHeight = " + height;
  }//toString
} //Rectangle
```

Två objekt av klassen Rectangle

Låt oss skapa två objekt av klassen Rectangle

```
Rectangle rec1 = new Rectangle(5.0, 5.5);
Rectangle rec2 = new Rectangle(10.0, 20.0);
```

Scenariot vi får kan illustreras med nedanstående figur.



Olika slag av metoder

Metoderna i en klass kan klassificeras enligt:

- *konstruktor* för att initiera tillstånden hos objektet som skapas

  Rectangle()
  Rectangle(**double** w, **double** h)

- ***avläsare*** (***accessor***, ***get-metoder***) för att avläsa värdet hos instansvariabler eller klassvariabler, dvs avläsa tillstånd hos ett objekt

  double getWidth()
  double getHeight()

- ***omformare*** (***mutator***, ***set-metoder***) som manipulerar instansvariabler eller klassvariabler, dvs ändra tillstånd hos ett objekt eller hos klassen

  void setWidth(**double** w)
  void setHeight(**double** h)

- *operation* som använder instansvariablerna för att göra beräkningar

  double getArea()
  double getPerimeter()
  String toString()

Metoder

En *instansmetod* anropas genom att ange namnet på objektet som skall anropas, följt av en punkt '.', följt av metodens namn och metodens argumentlista.

objektnamn.metodnamn(argumentlista)

En klassmetod anropas genom att ange klassens namn, följt av en punkt '.', följt av metodens namn och argumentlista.

Klassnamn.metodnamn(argumentlista)

Exempel:

```
Scanner data = new Scanner("12.0  5.5");
double width = data.nextDouble();
double height = data.nextDouble();
Random hazard = new Random();
int randomHeight = hazard.nextInt(100);
Rectangle firstRec = new Rectangle();
Rectangle secondRec = new Rectangle(width, height);
firstRec.setHeight(randomHeight);
double theWidth = secondRec.getWidth();
```

Det reserverade ordet this

Varje objekt har en referens *till sig själv*. Denna är dock inte "synlig" utan nås via det reserverade ordet **this**. I klassen Rectangle har metoden setWidth följande utseende:

```
public void setWidth(double w) {
    width = w;
} //setWidth
```

Men kan även skriva:

```
public void setWidth(double w) {
    this.width = w;
} //setWidth
```

```
public class Rectangle {
  private double width;
  . . .
  public Rectangle() {
    width = 0;
    height = 0;
  }
  public void setWidth(double w) {
    this.width = w;
  }//getWidht
  . . .
} //Rectangle
```

Det reserverade ordet this

Att referera till **this** är nödvändigt om instansvariabeln och den formella parametern har samma namn som i exemplet nedan:

```
public void setWidth(double width) {
    this.width = width;
} //setWidth
```

Anledningen till detta är att vi har en *namnkonflikt* och då gäller *närhetsprincipen*, vilket i detta fall betyder att width avser den aktuella parametern.

Att ha samma namn på instansvariabler och på formella parametrar till metoder är vanligt förekommande (width säger ju mer än w om vad parametern avser).

```
public class Rectangle {
   private double width;
   . . .
   public Rectangle() {
      width = 0;
      height = 0;
   }
   public void setWidth(double width) {
      this.width = width;
   }//getWidht
   . . .
} //Rectangle
```

Bättre variant av klass Rectangle

```
public class Rectangle {
   private double width;      //instansvariabel
   private double height;     //instansvariabel

   public Rectangle() {
      width = 0;
      height = 0;
   } //constructor

   public Rectangle(double width, double height) {
      this.width = width;
      this.height = height;
   }//constructor

   public double getWidth() {
      return width;
   }//getWidth

   public double getHeight() {
      return height;
   }//getHeight

   public void setWidth(double width) {
      this.width = width;
   } //setWidth

   public void setHeight(double height) {
      this.height = height;
   } //setHeight

   public double getArea() {
      return height * width;
   }//getArea

   public double getPerimeter() {
     return 2 * (height + width);
   } //getPerimeter

   public String toString() {
     return "Width = " + width + "\nHeight = " + height;
   }//toString
} //Rectangle
```

Synlighetsmodifierare

En metod har följande utseende:

```
modifierare resultattyp namn(parameterlista) {
   lokala deklarationer
   satser
}
```

En typ av modifierare är synlighetsmodifierare som anger metodens synlighet. Det finns fyra olika modifierare för synlighet

public	synlig för alla klasser
private	synlig endast i klassen själv
protected	synlig för klassen själv, klasser i samma paket och för subklasser
utelämnad	synlig för alla klasser i samma paket

Information hiding – dölja information

I klassen Rectangle är instansvariablerna width och height deklarerade som *privata*:

```
private double width;
private double height;
```

vilket innebär att de är *okända* för andra klasser.

För att kunna ta reda på i vilket tillstånd ett objekt befinner sig i måste det därför finnas publika åtkomstmetoder som returnerar instansvariablernas värden (*avläsare*):

```
public double getWidth()
public double getHeight()
```

Och för att kunna förändra tillståndet hos ett objekt måste det finnas publika metoder som ändra instansvariablernas värden (*modifierare*):

```
public void setWidth(double width)
public void setHeight(double height)
```

Information hiding – dölja information

Detta tillvägagångssätt att ha privata instansvariabler samt ha publika metoder för att avläsa och modifiera värdet av instansvariablerna kallas för *information hiding* och är en mycket viktig programmeringsprincip.

Information hiding ger fördelen att man *skiljer mellan objektets specifikation och objektets implementation*, dvs man behöver inte känna till objektets inre uppbyggnad för att kunna använda objektet.

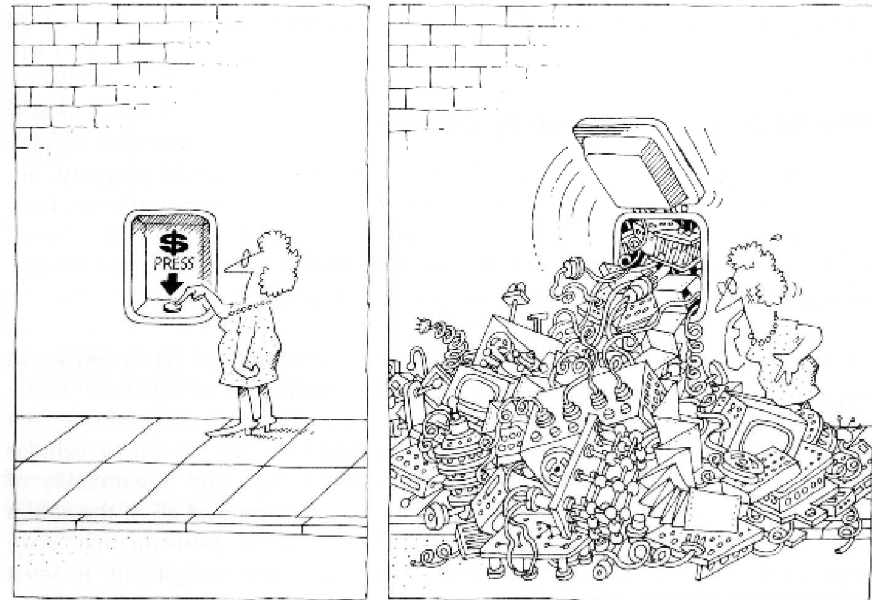
En annan fördel är att objektet kan isoleras mot fel i andra objekt.

Endast tillgång till gränssnitt

Även tillgång till implementationen

Information hiding – dölja information

Allt informationsutbyte mellan klient och server skall ske via ett väldefinierat *gränssnitt*, som utgörs av de publika metoderna som finns definierade för klassen som servern tillhör.



Keep it secret! Keep it safe!

Abstraktion

Abstraktion ger oss en möjlighet att hantera många olika fenomen/objekt på ett enhetligt sätt:

- För alla … gäller
- Inga .. har/kan …
- Samma …

Förenklar problemlösning, vi kan *bortse* från specialfall

- Gör tillvaron greppbar

Abstraktioner i kursen

- Metoder, klasser
- Överlagring (overload), kan använda samma namn för olika metoder (även flera konstruktorer)
- `implements`, alla objekt som implementerar gränssnittet (`interface`) kan behandlas på samma sätt (alla objekt har de metoder som anges i gränssnittet), t.e. List
- `extends`, alla supklasser kan behandlas som superklassens, de kan minst lika mycket (ärver kod), t.e. Svensk, Holländare
- Överkuggning (`override`), alla objekten vet själva hur de skall bete sig, vilken betos som anropas, avgörs av typen för objektet
- Samlingar, alla samling kan hantera vilka objekt som helst, vi anger typen med t.ex. List<String> eller List<Tile>, DiceCup

Lambda uttryck

- Java fick äntligen lambda uttryck i version 8
- Anonymous funktion: en funktion utan namn
- Higher-order metoder: en metod som tar en funktion som argument
- Demo!