F6 – Objektorienterad design

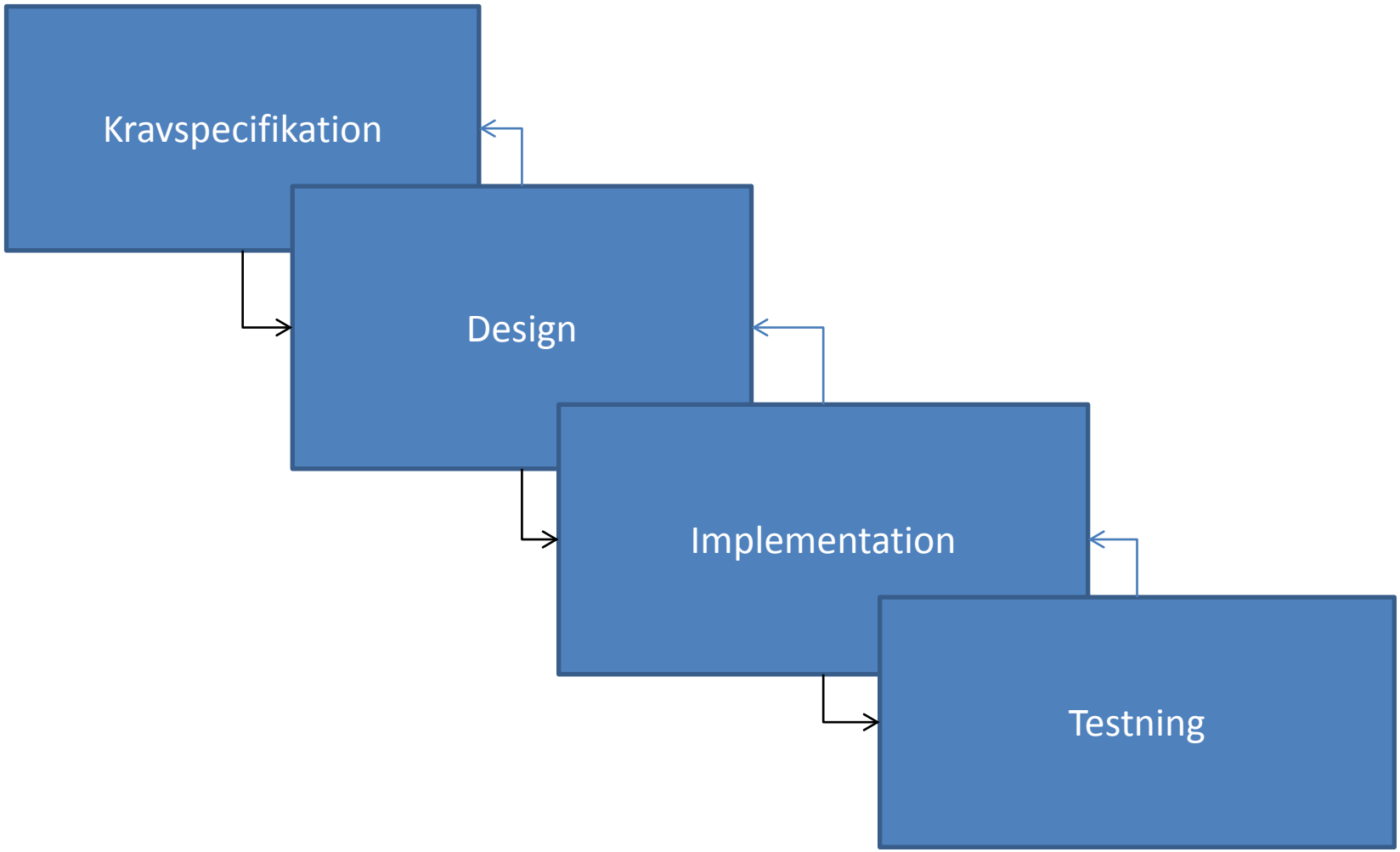
ID1004 – Objektorienterad programmering

Fredrik Kilander fki@kth.se

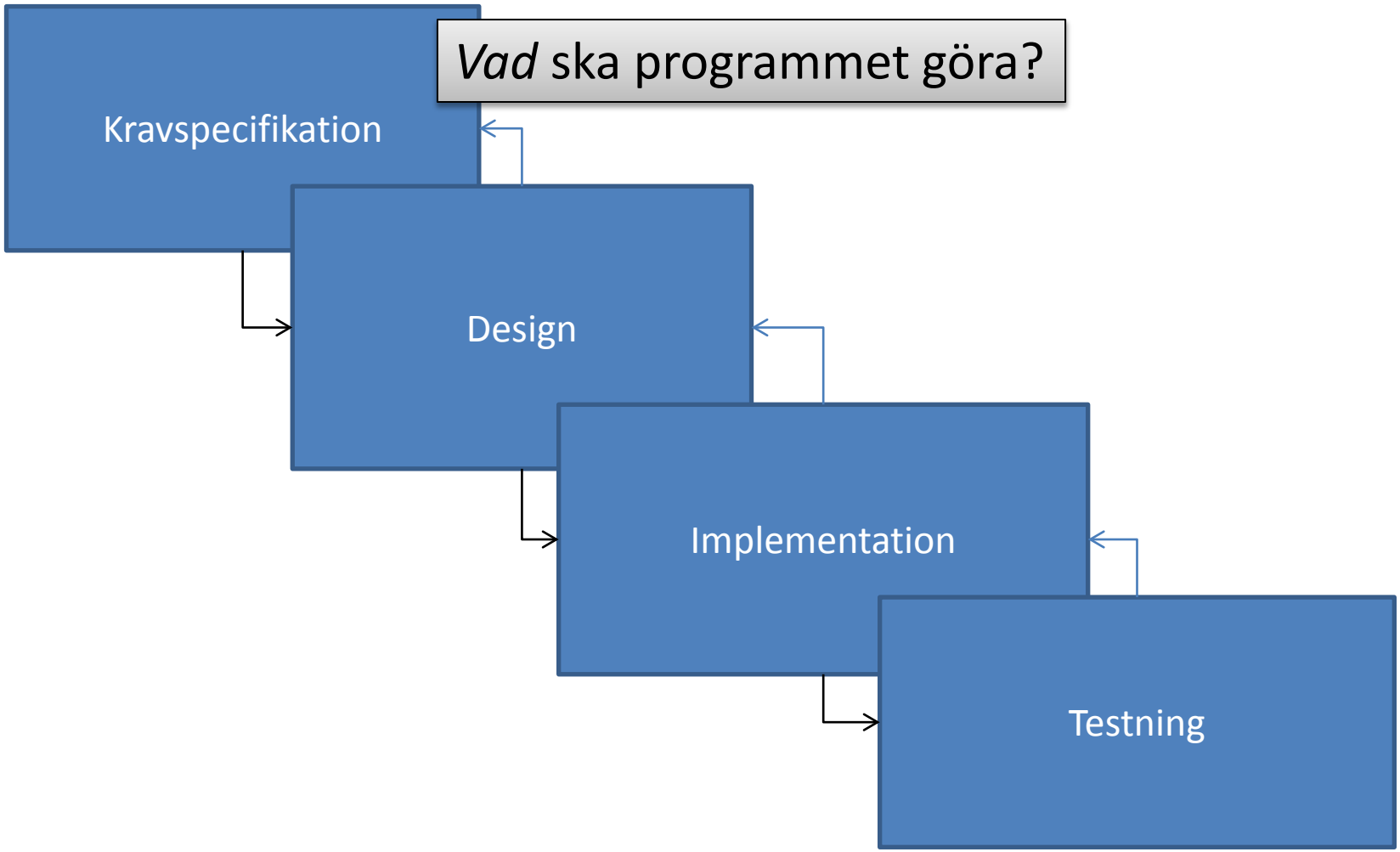
långa ord

AKTIVITETER I PROGRAMVARUUTVECKLING

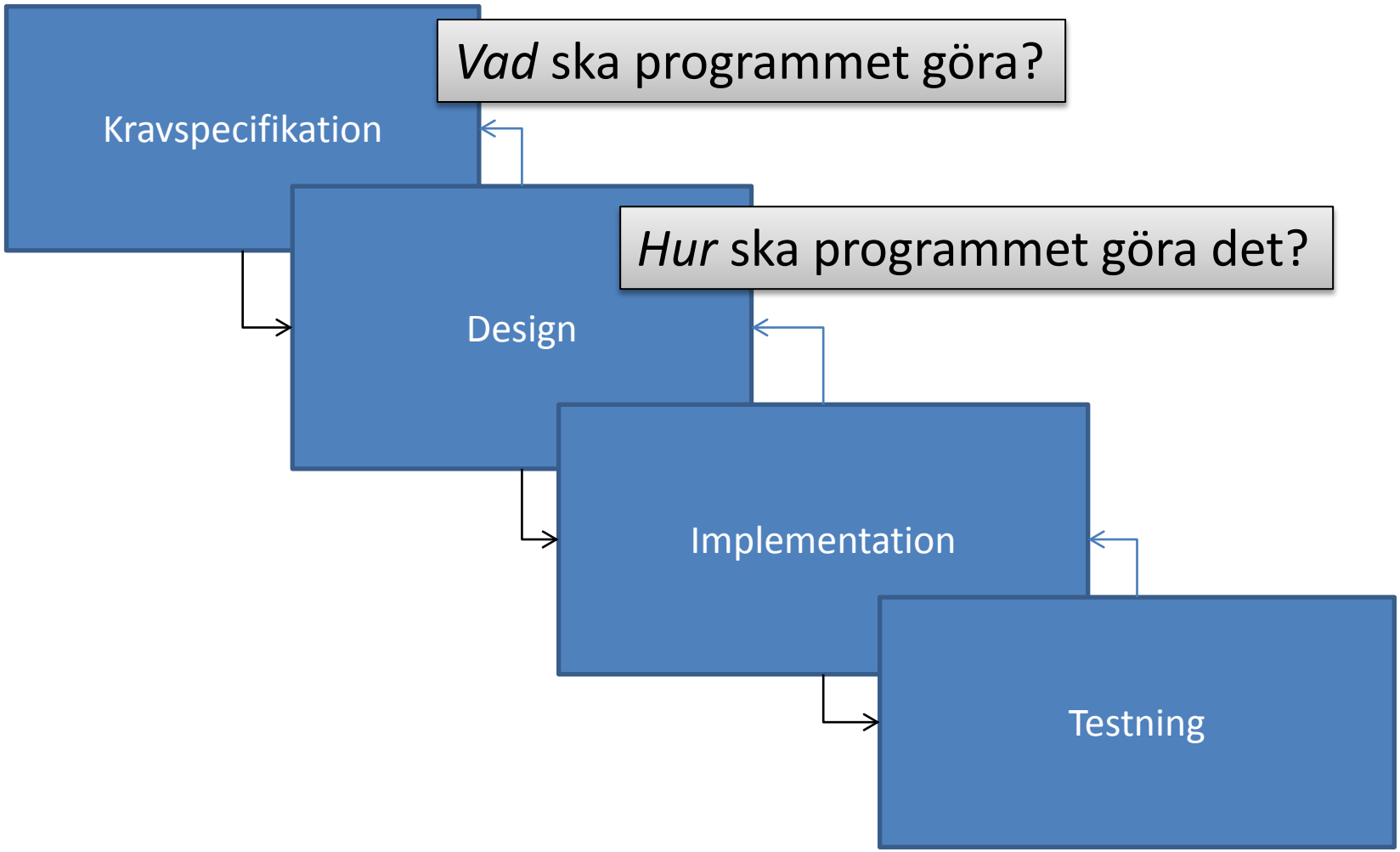
Iterativ utveckling



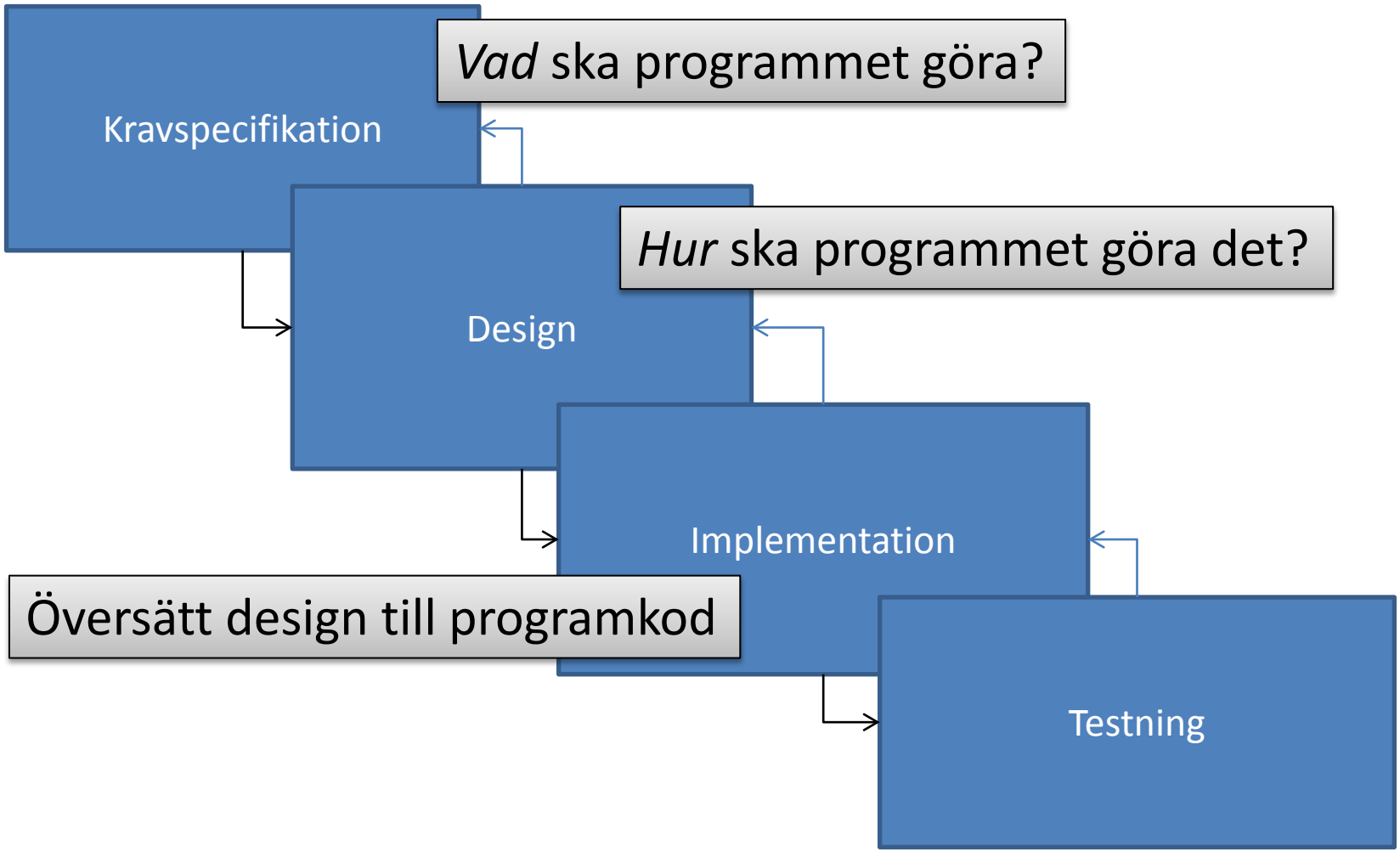
Iterativ utveckling



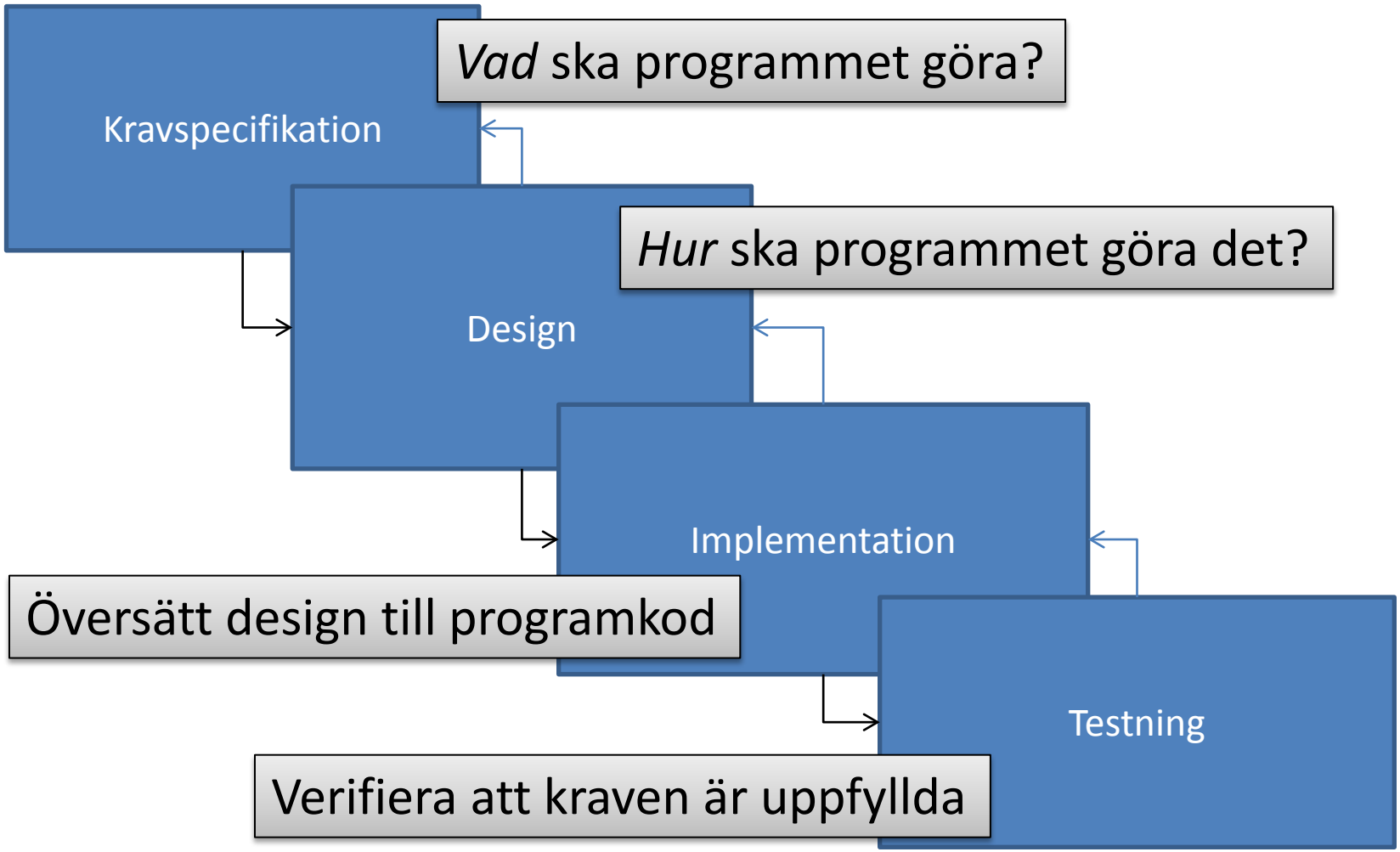
Iterativ utveckling



Iterativ utveckling

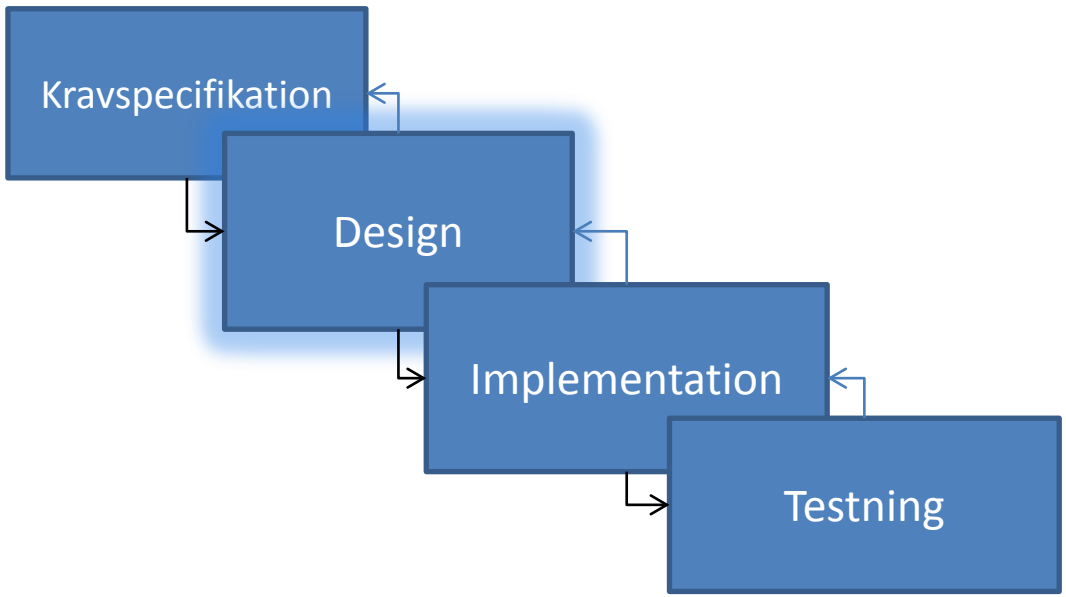


Iterativ utveckling



Objektorienterad design

- Designfasen är den kreativa fasen
- Implementation är att bygga efter ritning



Objektorienterad design

- Metodansats: substantiv i kravspecifikationen

Användaren måste kunna ange produkt med hjälp av dess huvudsakliga egenskaper, inklusive dess namn och produktnummer. Om streckkoden inte stämmer med produkten ska ett fel genereras i meddelandefönstret och loggas i felloggen. Sammanfattningsrapporten över alla transaktioner ska vara strukturerad enligt spec 7.A.

Objektorienterad design

- Metodansats: substantiv i kravspecifikationen

Användaren måste kunna ange produkt med hjälp av dess huvudsakliga egenskaper, inklusive dess namn och produktnummer. Om streckkoden inte stämmer med produkten ska ett fel genereras i meddelandefönstret och loggas i felloggen. Sammanfattningsrapporten över alla transaktioner ska vara strukturerad enligt spec 7.A.

Substantiven i kravspecifikationen

användare

produkt

fel

produktegenskaper

fellogg

namn

produktnummer

streckkod

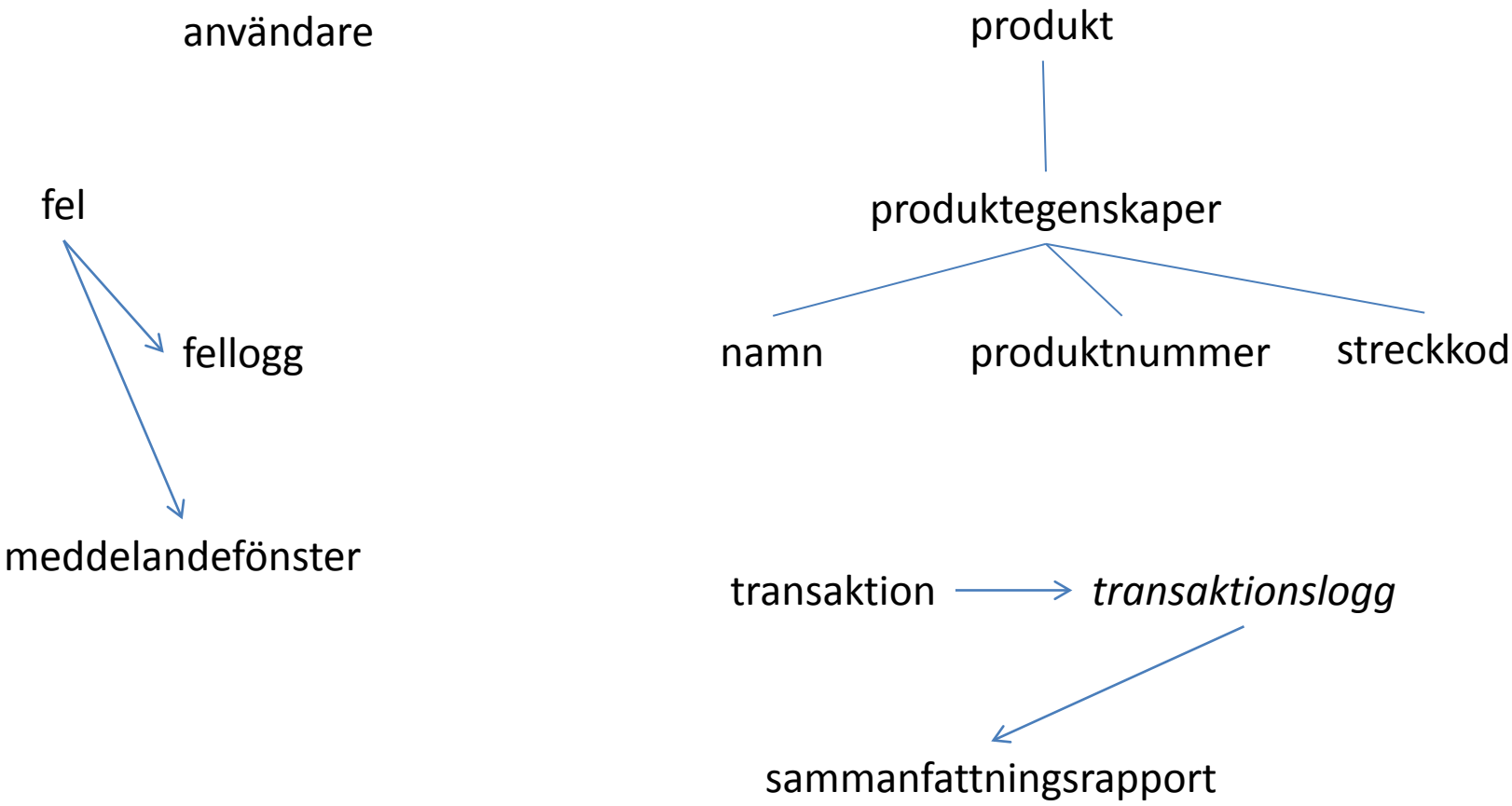
meddelandefönster

transaktion

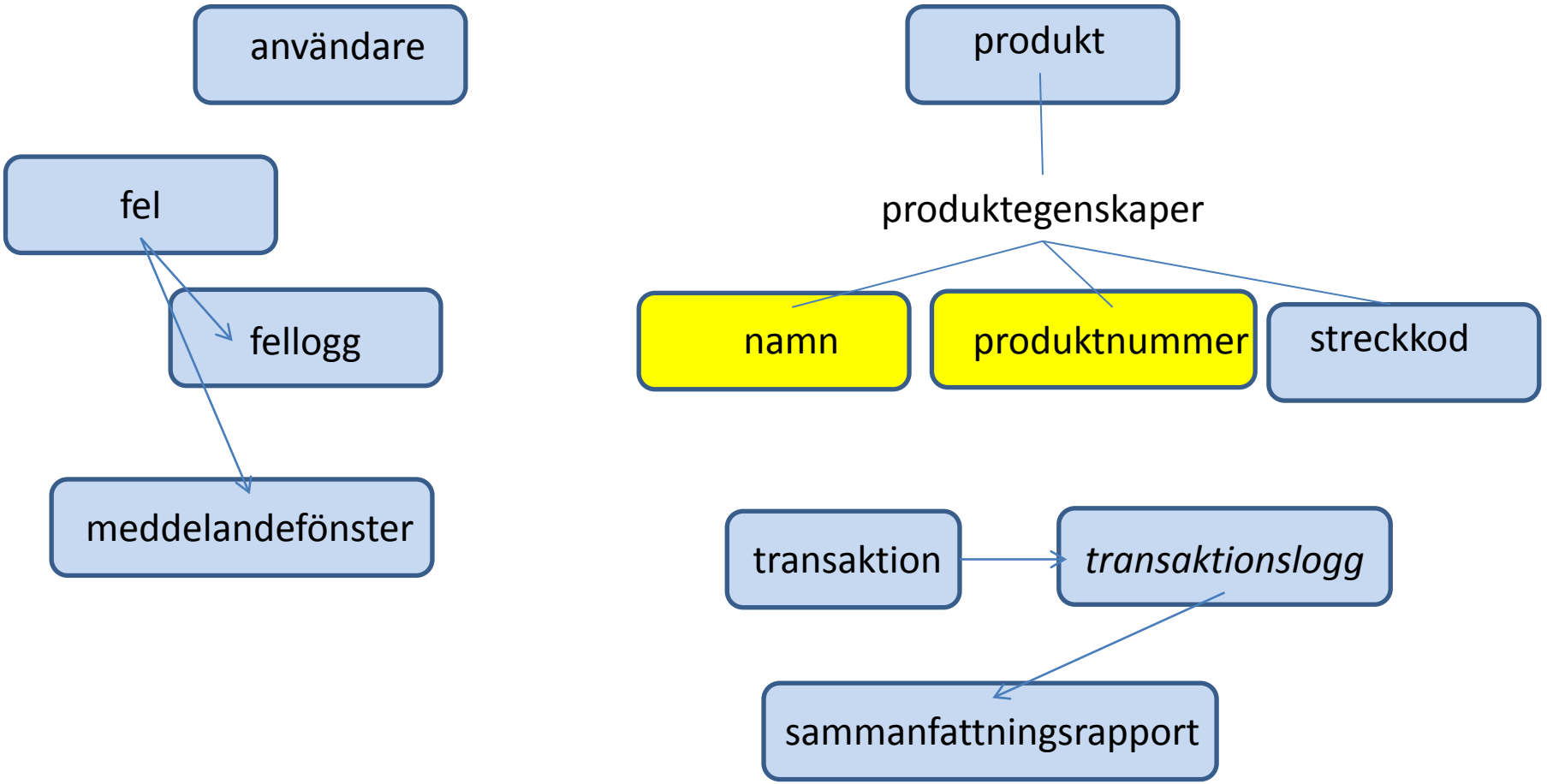
transaktionslogg

sammanfattningsrapport

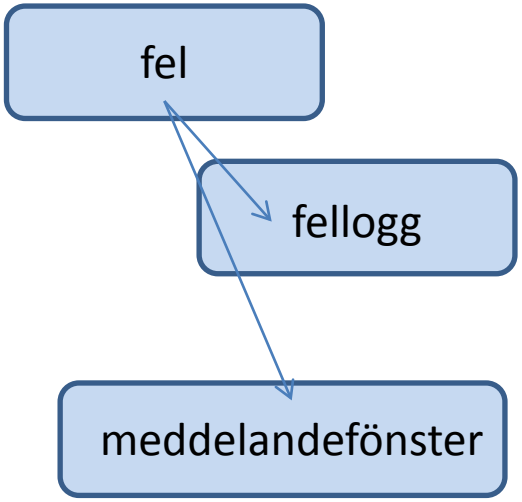
Substantiven – tentativa relationer



Tänkbara klasser och attribut



Tänkbara beteenden

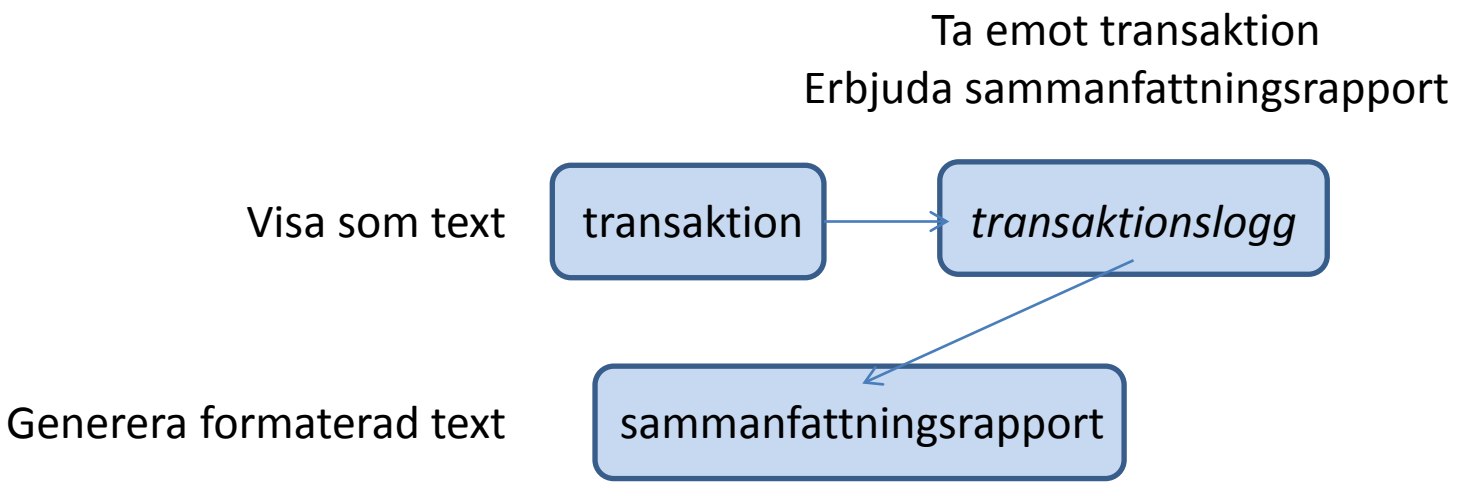


Presentera som text

Ta emot och lagra fel

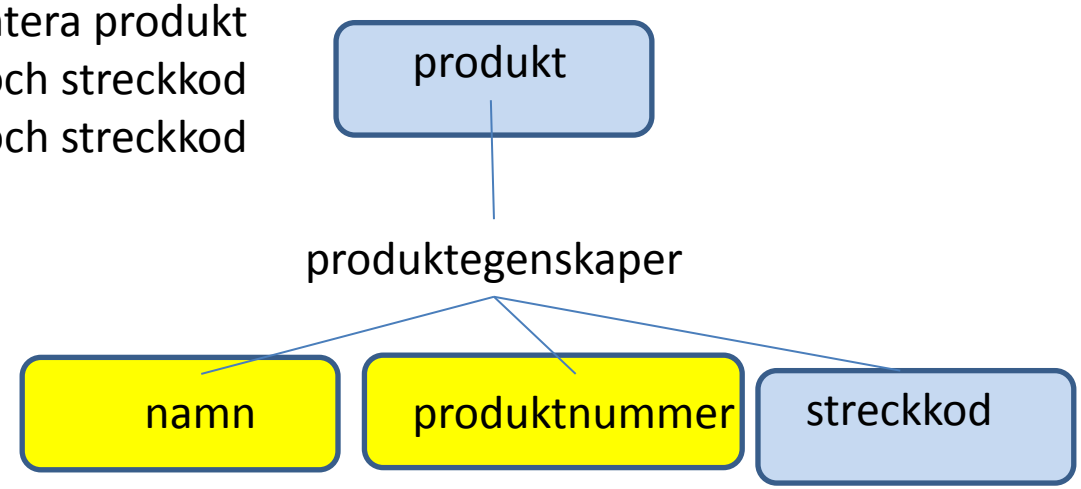
Visa fel

Tänkbara beteenden



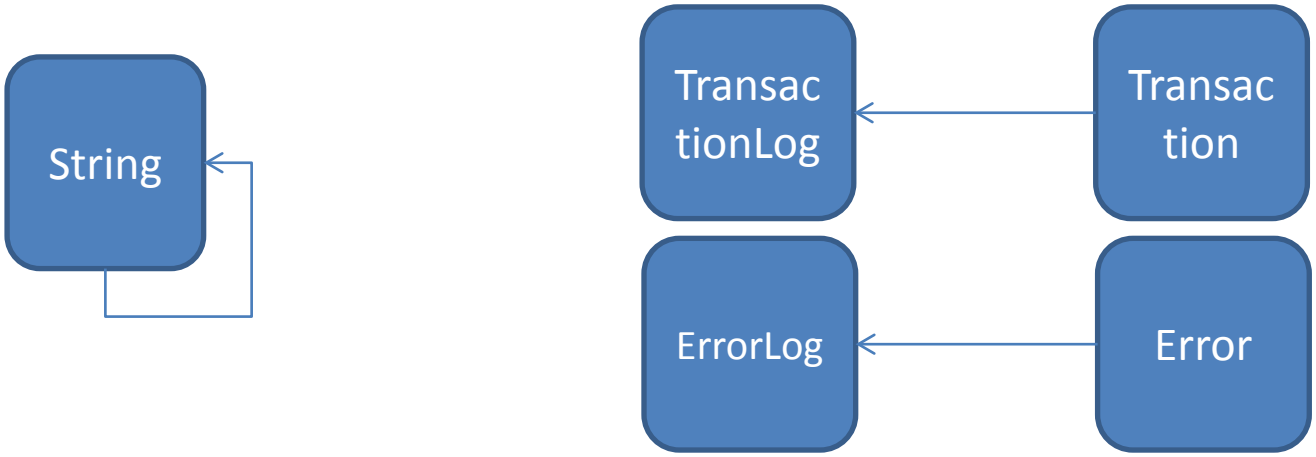
Tänkbara beteenden

Presentera produkt
Erbjuda namn, produktnummer och streckkod
Jämföra produktnummer och streckkod



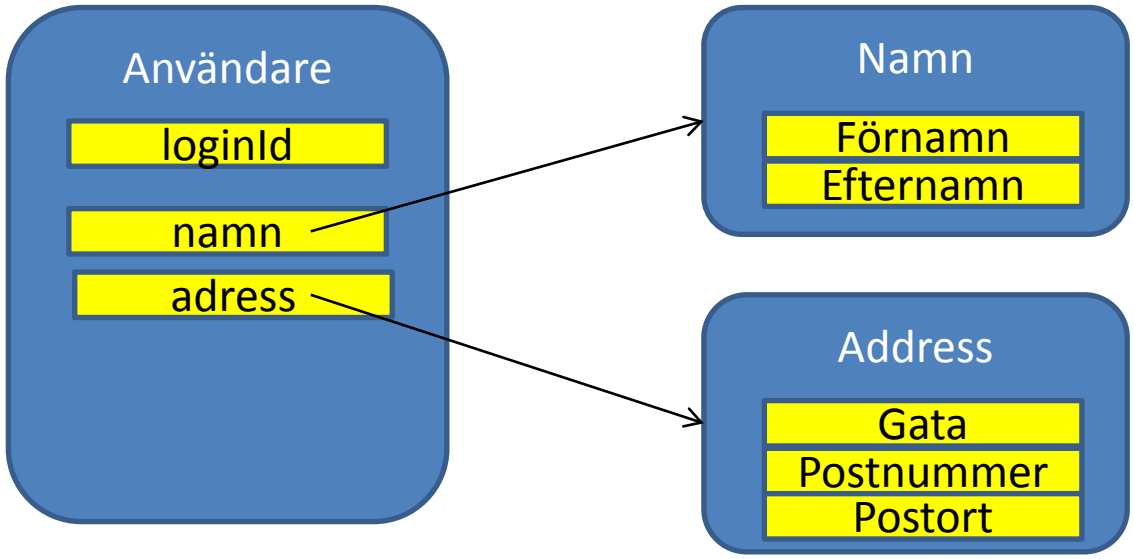
Design – beroenden mellan klasser

- Beroenden mellan objekt i samma klass
- **`str3 = str1.concat(str2)`**
- Beroenden mellan objekt i olika klasser
- **`TransactionLog.add(Transaction t)`**
- **`ErrorLog.add(Error e)`**



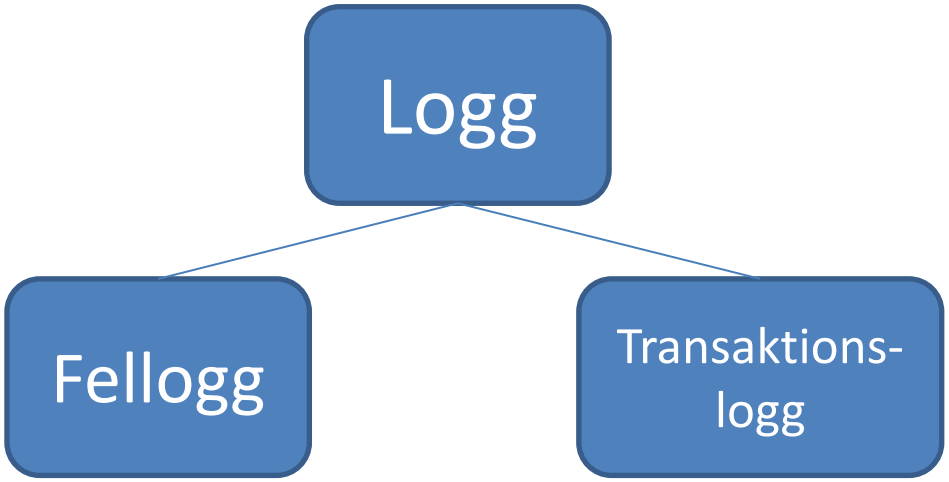
Design - aggregering

- Aggregerade objekt byggs upp med hjälp av andra objekt



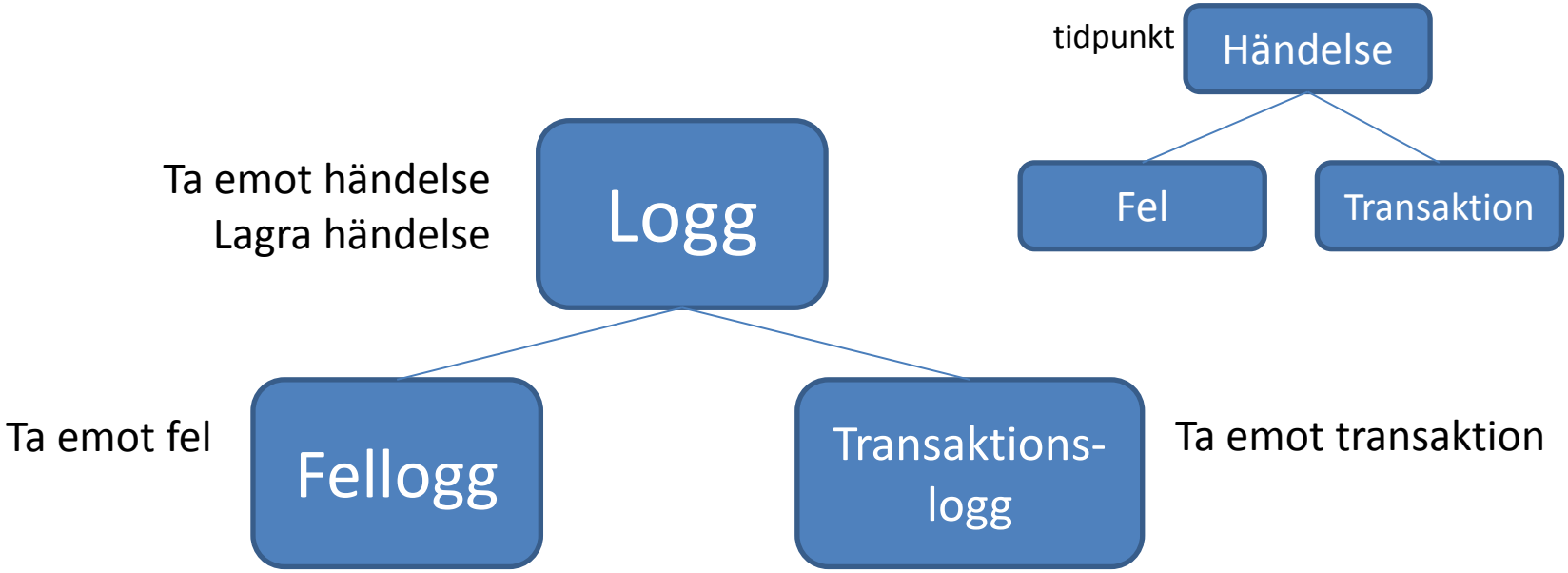
Design – gemensamma egenskaper

- Felloggen och transaktionsloggen är båda loggar
- Båda lagrar information om sådant som hänt
- Hur mycket har de gemensamt?



Design – gemensamma egenskaper

- Felloggen och transaktionsloggen är båda loggar
- Båda lagrar information om sådant som hänt
- Hur mycket har de gemensamt?



mellantryne

INTERFACE-KLASSER

Interface-klasser

- Interface-klasser innehåller
  - konstanter
  - abstrakta metoder
- Abstrakta metoder har signatur och returtyp, men ingen kod
- Ett interface specificerar ett *beteende*

Signatur: en metods klass, namn, och ordningsföljden på parametertyperna

Interface - exempel

```
package java.lang;
public interface Runnable {
  public void run ();
}
```

```
public class MyClass implements Runnable {
  …
  public void run() {
    …
  }
  …
}
```

Interface-klasser

```
public interface Comparable<T> {
  public int compareTo (T o);
}
```

En klass som implementerar Comparable, lovar att den implementerar alla metoder i Comparable.

```
public class Die implements Comparable<Die> {
  ...
  public int compareTo (Die otherDie) {
    return otherDie.getValue() – this.getValue();
  }
  ...
}
```

Interface-klasser

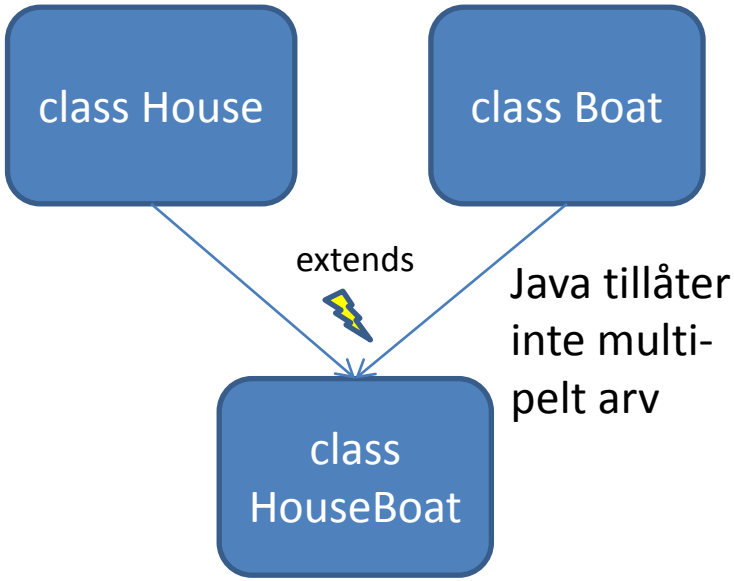
```
public interface Iterator<E> {
   public boolean hasNext();
   public E next();
   public void remove();
}
```

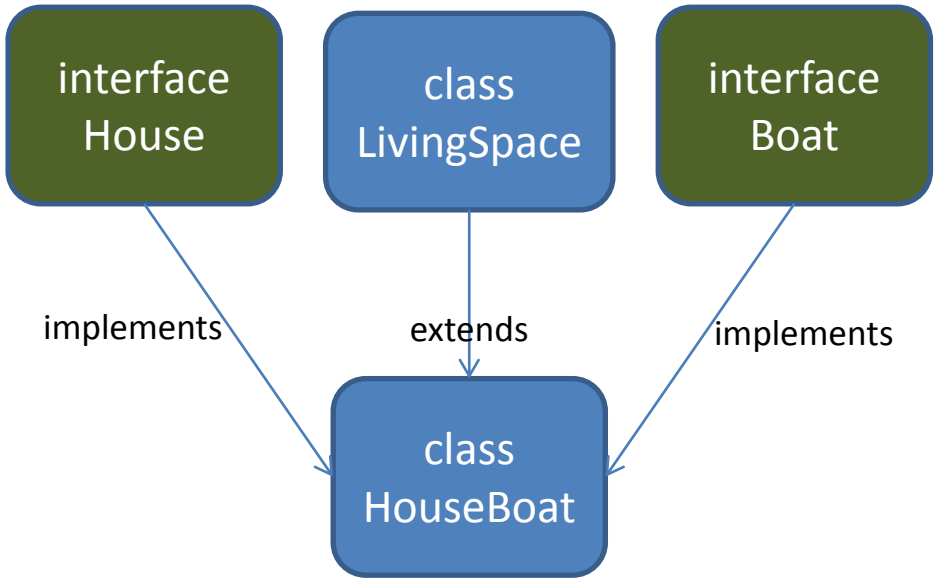
Interface används ofta

- En klass kan implementera många interface
- En klass som implementerar ett interface utlovar ett väldefinierat beteende
- För att använda objektet behöver man bara känna till interfacet

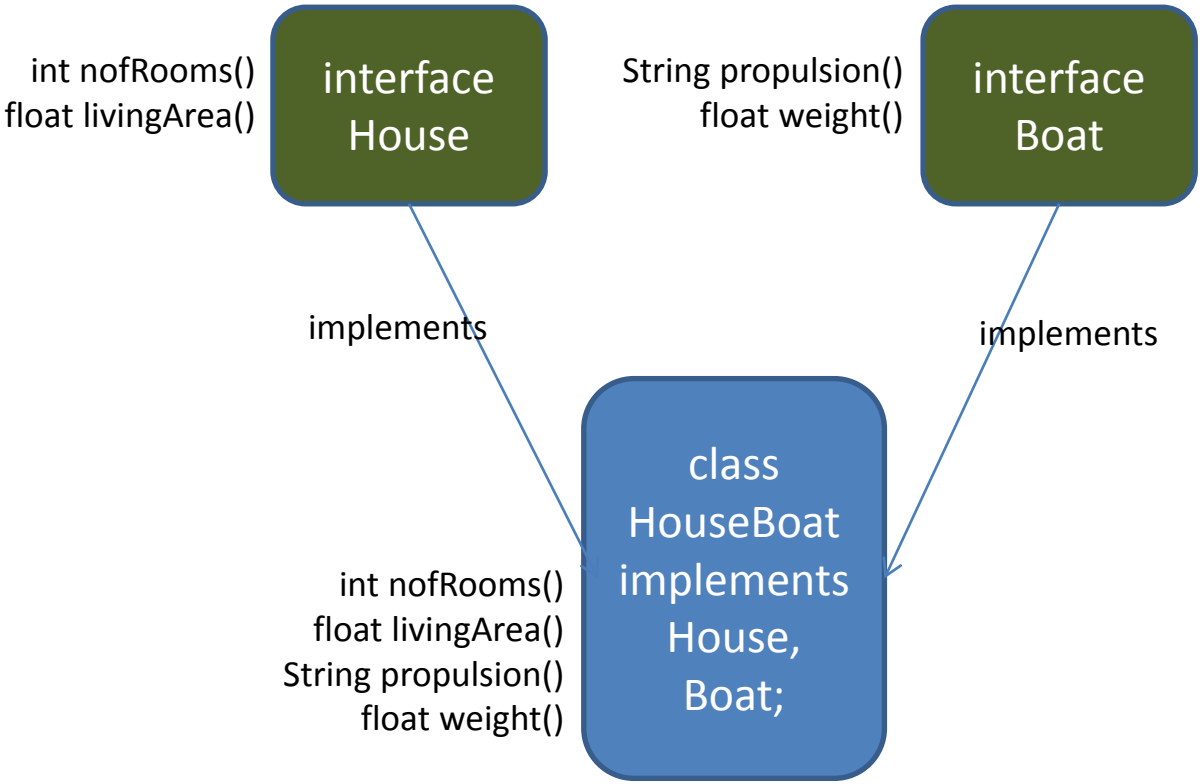
- Interface löser problem som arv inte kan lösa

Interface-klasser (exempel)

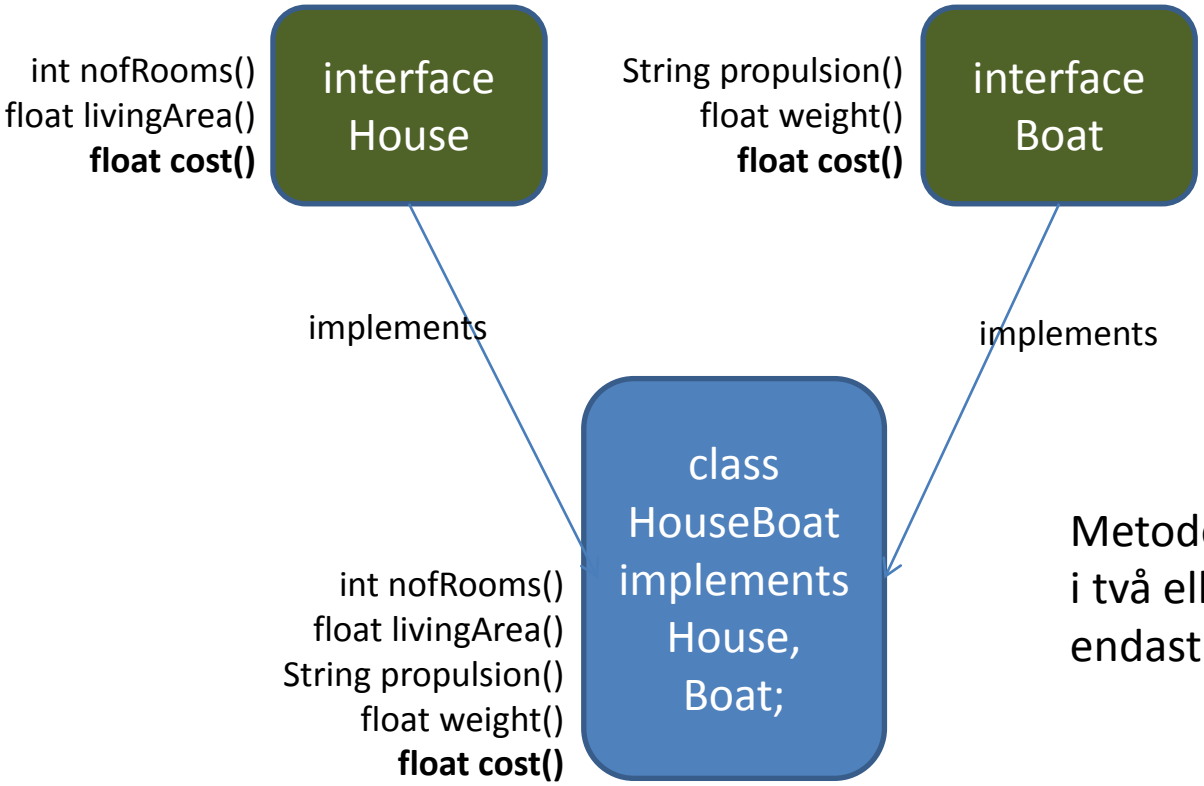




Interface-klasser (exempel)

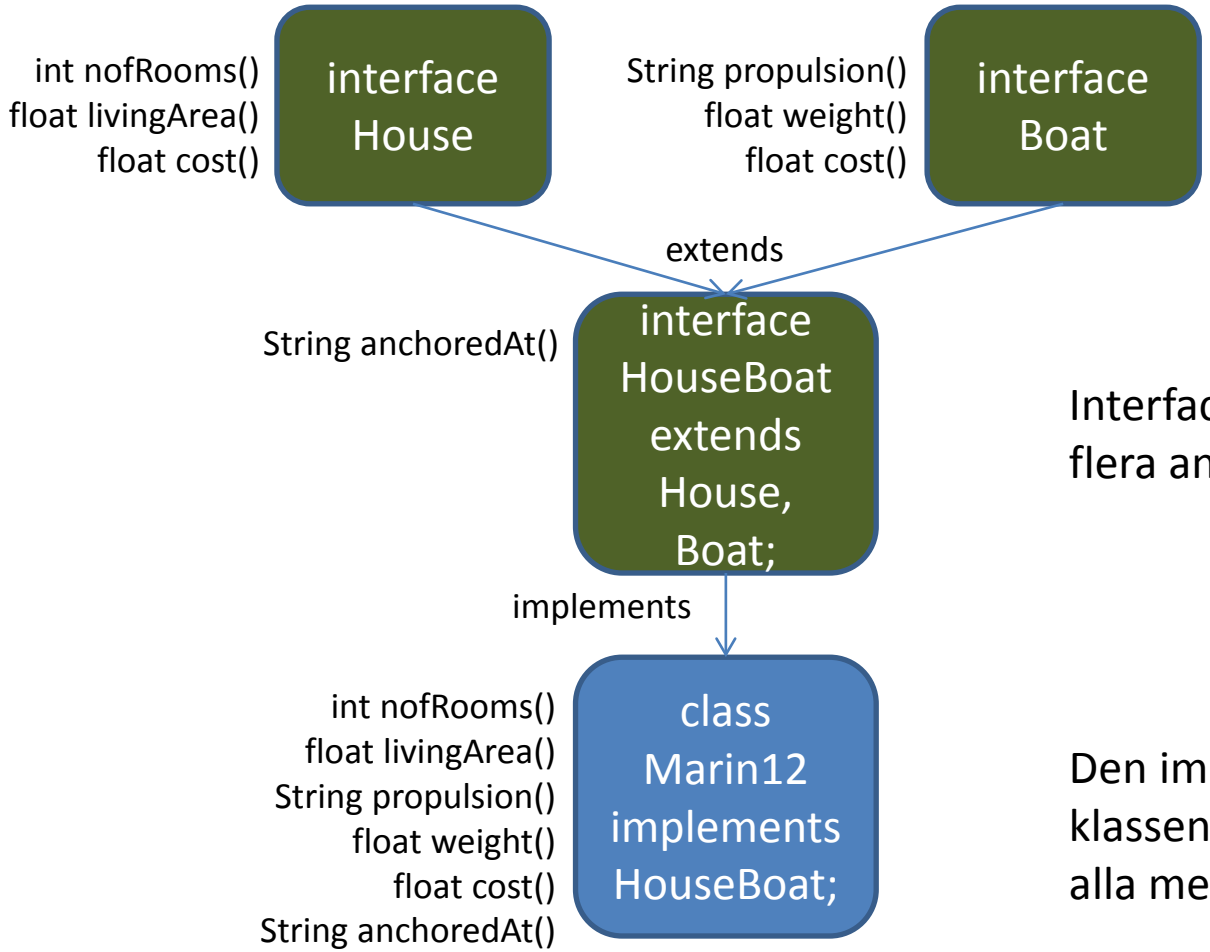


Interface-klasser (exempel)



Metoder som är identiska i två eller fler interface får endast en implementation.

Interface-klasser (exempel)



Interface kan ärva från flera andra interface

Den implementerande klassen måste implementera alla metoder

Interface – till sist

- Ett interface är också en datatyp:
  - **`protected House huset;`**
- Variabeln huset kan referera till alla objekt som implementerar interface House.
- Metoder kan returnera datatypen:
  - **`public House findHouse(String address)`**
    `...`

hjälp!

HJÄLPMETODER

Metoder och hjälpmetoder

- Hjälpmetoder förenklar koden
- Undviker repetition och redundans
- Bryter ned i enklare delar

Metoder och hjälpmetoder

```
int x, y, z;
Scanner scan = new Scanner(System.in);
...
boolean argOk = false;
while (!argOk) {
  System.out.print(”Enter x : ”);
  x = scan.nextInt();
  if ((X_MIN <= x) && (x < X_MAX)) {
    argOk = true;
  }
  else {
    System.out.println(”x is out of range”);
  }
}
// Repeat for y and z
```

Metoder och hjälpmetoder

```
int x, y, z;
Scanner scan = new Scanner(System.in);
...
boolean argOk = false;
while (!argOk) {
  System.out.print(”Enter x : ”);  // anmoda användaren
  x = scan.nextInt();              // läs in värde
  if ((X_MIN <= x) && (x < X_MAX)) {
    argOk = true;                  // värdet är godkänt
  }
  else {
    System.out.println(”x is out of range”); // felmeddelande
  }
}
// Repeat for y and z
```

Metoder och hjälpmetoder

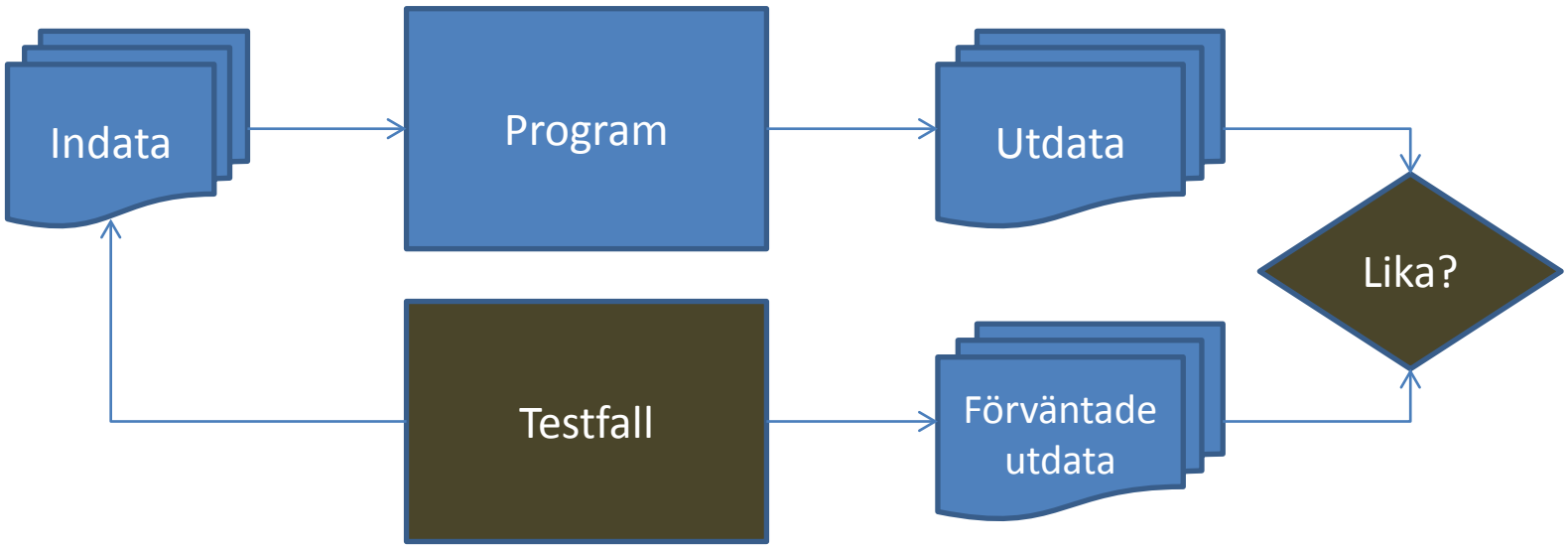
```
private int getCoordinate(String name, Scanner scan,
                          int min, int max) {
  while (true) {
    System.out.print(”Enter ” + name + ” : ”);
    int value = scan.nextInt();
    if ((min <= value) && (value < max)) return value;
    System.out.println(name + ” is out of range”);
  }
}

  int x, y, z;
  Scanner scan = new Scanner(System.in);
  ...
  x = getCoordinate(”x”, scan, X_MIN, X_MAX);
  y = getCoordinate(”y”, scan, Y_MIN, Y_MAX);
  z = getCoordinate(”z”, scan, Z_MIN, Z_MAX);
```

”failure is always an option”

TESTNING

Testning

- Man testar för hitta fel
- Felfritt test == programmet klarar just det testet
- Det brukar alltid finnas några fel kvar



Testning

- Ofta använder man flera olika tester
- Ibland leder en justering till nya fel
- *Regression testing* kör om hela testbatteriet efter varje justering
- Automatiserad testning underlättar

Test 1 Test 2 Test 3 Test 4 Test 5 ...

Testning – kodgenomgång

- Flera personer läser källkoden tillsammans
- Tar tid
- Kan ge mycket bra resultat
- Identifiera problemen – inte lösa dem

Defekttestning

- Skapa test med syfte att hitta fel
- Black-box testing
  - Programmet är en svart låda
  - Känd input, förväntad output
- White-box (glass-box) testing
  - Testet är utformat för att gå alla vägar genom koden

Testning – ekvivalenta kategorier

- Indata delas in i mängder
- Varje element i en mängd antas ha lika stor chans att hitta ett visst fel
- Exempel: beräkna kvadratroten
  - double sqrt(double u)
  - Två mängder: positiva tal och negativa tal
  - Testa med några positiva tal
  - Testa med några negativa tal

Testning i praktiken

- Välj indata nära gränserna

Testning i praktiken

- Välj indata nära gränserna
- Om en funktion är definierad f(m)...f(n), m<n
  - f(m-1) : omfångsfel väntas
  - f(m) : rätt svar väntas
  - f((m+n)/2) : rätt svar väntas
  - f(n) : rätt svar väntas
  - f(n+1) : omfångsfel väntas

Testning i praktiken

- För ett enklare program

- Be någon som aldrig sett programmet köra det
  - ”Varför kan man inte mata in noll här?”
  - ”Nä, det meddelandet såg jag inte”
  - ”Jag fattar inte vad jag ska göra”

...

SLUT PÅ BILDER

Testning i praktiken

- Specialfall bildar egna ekvivalensklasser
- x^y : x^0, x^y

```
int power(int x, int y) {
  if (y == 0) {
    return 1;
  }
  else {
    return x * power(x, y-1);
  }
}
```