Innehåll

- OOP snabbintroduktion
- Programvaruutveckling och programmering
- Datatyper
- Uttryck
- Satser
- Arv, polymorfi och dynamisk bindning
- Att organisera Javakod
- Metodik och klassdesign
- Design (CRC)
- Fält
- Undantag
- In-/utmatning och filer
- Grafik
- Rekursion
- GUI:s
- Applets vs applikationer
- Interfaces och sortering
- Noggrannheten i beräkningar

En applikation är ett fristående program

- En Java applikation måste innehåller en `main()` metod
- I `main()` metoden startas exekveringen
- `Main()` metoden måste se ut på ett särskilt sätt

```
public class ExampleApplication
{
   public static void main (String[] argv)
   {
      // Deklarationer och satser som i
      // vilken metod som helst
      …
   }
   …
}
```


En applet är inbäddad i en websida

```
<html>
<title>En sida med en applet</title>
<applet
   code=AppletSubklass.class
   width=bredden_i_pixel height=höjden_i_pixel>
</applet>
</html>
```

- Applet klassen måste ärva från `java.awt.Applet` (direkt eller indirekt)
- Se `Einstein.java` och `Fahrenheit.java`

En applet är inbäddad i en websida

- Browsern (Netscape, Explorer etc)
  - Laddar en applet över nätet när den hittar en `<applet>`-tag
  - Tillhandahåller en grafisk miljö
  - Skapar ett objekt av appletens klass
  - Tillåter inte vissa operationer av säkerhetsskäl
- Exempel:

```
<html>
<title>En applet sida</title>
<applet
   code=AppletSubclass.class
   width=anInt height=anInt>
</applet>
</html>
```

- Se också

En applets livscykel

- Applet objektet skapas av browsern när den finner en `<applet>`-tag (alt. appletviewer:n)
- Inititialiseringen
  - Endast en gång när appleten laddas
  - Motsvarar konstruktorn
- Startas
  - Varje gång sidan visas
- Stoppas
  - När sidan lämnas
- Slutstädas
  - När sidan tas bort ur minnet

Applet
... + init ( ) + start ( ) + stop ( ) + destroy ( ) + paint (Graphics g) + play (URL source) + resize (int w, int h) ...

Vad applets inte kan/ får

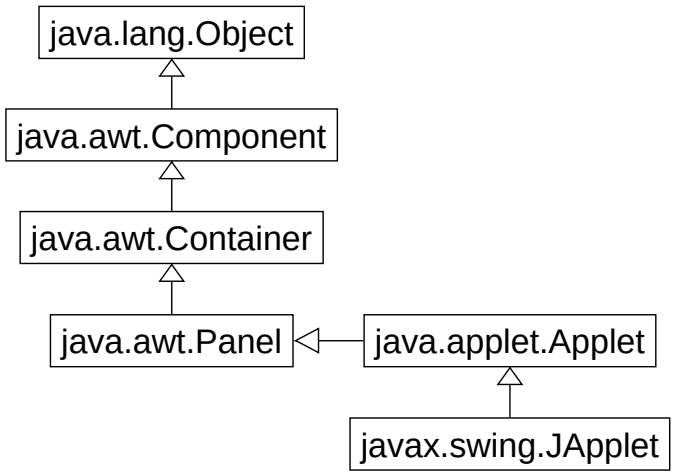
- Ladda bibliotek eller “native methods”
- Läsa och skriva filer på klienten
- Göra nätverksanslutningar annat än till sin “egen” server
- Starta program på klienten
- Läsa vissa systemegenskaper

Vad applets kan som inte applications kan

- Spela ljudfiler
- Visa HTML-dokument
- Prata med andra applets på samma sida

- Applets som laddats “lokalt” har inga begränsningar

Applet



Innehåll

- OOP snabbintroduktion
- Programvaruutveckling och programmering
- Datatyper
- Uttryck
- Satser
- Arv, polymorfi och dynamisk bindning
- Att organisera Javakod
- Metodik och klassdesign
- Design (CRC)
- Fält
- Undantag
- In-/utmatning och filer
- Grafik
- Rekursion
- GUI:s
- Applets vs applikationer
- Interfaces och sortering
- Noggrannheten i beräkningar

Att organisera Javakod

- Abstrakta klasser (`abstract`)
  - När metoderna i lämpliga superklasser blir för allmänna för att kunna implementeras
  - Se Föreläsning 7
- Interfaces (`interface`)
  - När abstrakta klasser ska passa in i flera arvshierarkier

```
public interface ActionListener extends EventListener
{
   void actionPerformed (ActionEvent e);
}
```

- Paket (`package`)
  - När klasserna ska struktureras i olika filkataloger
  - Inte här

Interfaces 1

- Ett Java `interface` är en mängd konstanter och abstrakta metoder
- Ett interface specificerar “krav” på de klasser som implementerar interfacet
- En klass som implementerar ett interface måste implementera **alla** dess metoder
- Varje `interface`-implementation är en ny “version” av interfacet

```
class klassnamn implements interfacenamn
{
   ...
}
```

OBS! Det får finnas flera

Interfaces 2

- En klass kan ärva från en klass och dessutom implementera ett eller flera interface
- Interface konstanterna är tillgängliga i den implementerande klassen
- En interface kan “ärva” från ett eller flera interface (m.h.a. `extends`)

Interfaces 3

- Interface är inga klasser och kan inte instantieras
- Varje interface definierar dock en klasstyp
- ➔ Interface liknar abstrakt superklass
  - ➔ Man får definiera variabler av interface typ
  - ➔ Variablerna får referera till objekt av implementerande klasser

```
ActionListener aL =
   new ConcreteActionListener();
```

- Interface-konceptet har många likheter med (multipel) arv, dock utan dess nackdelar
- Multipel arv finns inte i Java

Sortering

- Många olika algoritmer
- Ingen är bäst sämst i alla lägen

- Här: Bara två enkla algoritmer
- Mera i DoA

Urvalssortering

- Generell idé:
  - Välj ett element
  - Sätt in på rätt plats
  - Repetera för alla element

- Förfining:
  - Sök minsta elementet
  - Sätt in på första positionen
  - Sök näst minsta element
  - Sätt in på andra positionen
  - ...
  - Repetera tills alla element sitter rätt

Urvalssortering

- Exempel:

Urvalssortering--Javakod

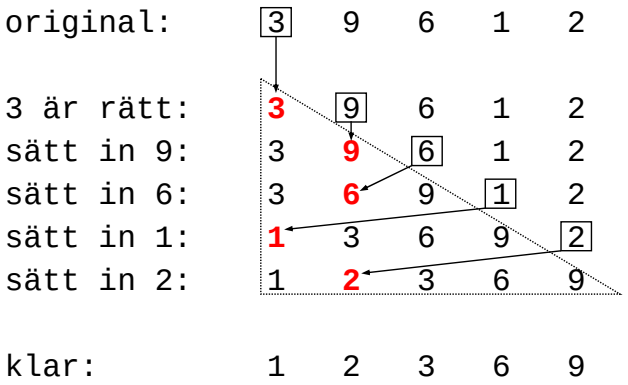
```
public static void sort (int[] numbers)
{
   int minIndex;   // index av aktuell minumum
   int temp;       // hjälpvariabel
   for (int i=0; i < numbers.length-1; i++)
   {
      minIndex = i;
      // sök minimum i delen som är kvar
      for (int j=i+1; j < numbers.length; j++)
         if (numbers[j] < numbers[minIndex])
            // nytt minimum hittad; spara dess index
            minIndex = j;
      // flytta minimumet till rätt position
      temp = numbers[minIndex];
      numbers[minIndex] = numbers[i];
      numbers[i] = temp;
   }
}
```


Insättningssortering

- Samma grundidé som urvalssortering:
  - Välj ett element
  - Sätt in på rätt plats
  - Repetera för alla element
- Ny förfining:
  - Sök inte efter minimum varje gång
  - Ta helt enkelt alla element i tur och ordning och sätt in dem på rätt plats i främre delen av fältet
  - Förflytta element som stod fel
  - Repetera tills alla element sitter rätt

Insättningssortering

- Exempel:



Insättningssortering--Javakod

```
public static void sort (int[] numbers)
{
   for (int i=1; i < numbers.length; i++)
   {
      int aktuell = numbers[i];
      int pos = i;

      // flytta större element till åt höger
      while (pos > 0 && numbers[pos-1] > aktuell)
      {
         numbers[pos] = numbers[pos-1];
         pos--;
      }

      // rätt position för aktuellt element hittad
      numbers[pos] = aktuell;
   }
}
```

En allmän objektsorterare

- Problem:
  - Objekt kan inte sorteras med “<”
  - Varje klass måste har en egen metod “sort”?
  - Strider mot målet återanvändning!
- Idé: Skriv en allmän objektsorterare
- Kraven läggs fast i ett Java interface:
  - Två objekt kan jämföras
  - Objekt kan sparas på en viss position (index)
  - Kolla upp objektet på en viss position
  - Kolla upp totala antalet objekt

Sortable interfacet

```
interface Sortable
{
   // returnera sant omm left > right
   boolean bigger (Object left, Object right);

   // returnera objektet på position `index´
   Object valueAt (int index);

   // spara `value´ på position `index´
   void setValue (Object value, int index);

   // returnera antalet element
   int length();
}
```


Sortable interfacet

- Nu kan interfacet användas för att skriva en allmän objektsorterare (här: insättningssortering)

```
public static void sort (Sortable items)
{
   for (int i=1; i < items.length(); i++)
   {
      Object aktuell = items.valueAt(i-1);
      int pos = i;
      while (pos > 0 &&
         items.bigger (items.valueAt(pos-1), aktuell)
      {
         items.setValue (items.valueAt(pos-1), pos);
         pos--;
      }
      items.setValue (aktuell, pos);
   }
}
```


Innehåll

Repetition: Hel- och flyttal

- De olika heltal och flyttal typer har olika storlek
- ➔ Kan ta upp olika värden med olik noggrannhet

Typ	Storlek	Minimum	Maximum
`byte`	8 bits	-128	127
`short`	16 bits	-32,768	32,767
`int`	32 bits	-2,147,483,648	2,147,483,647
`long`	64 bits	$< -9 \times 10^{18}$	$> 9 \times 10^{18}$
`float`	32 bits	$+/- 3.4 \times 10^{38}$	med 7 signifikanta siffror
`double`	64 bits	$+/- 1.7 \times 10^{308}$	med 15 signifikanta siffror

Många beräkningar är approximationer

- Val av modell
  - idealiseringar
- Fel i indata
  - mätvärden med begränsad noggrannhet
- Avrundningsfel
  - ändlig aritmetik
- Trunkeringsfel
  - ett oändligt värde ersätts med en ändlig

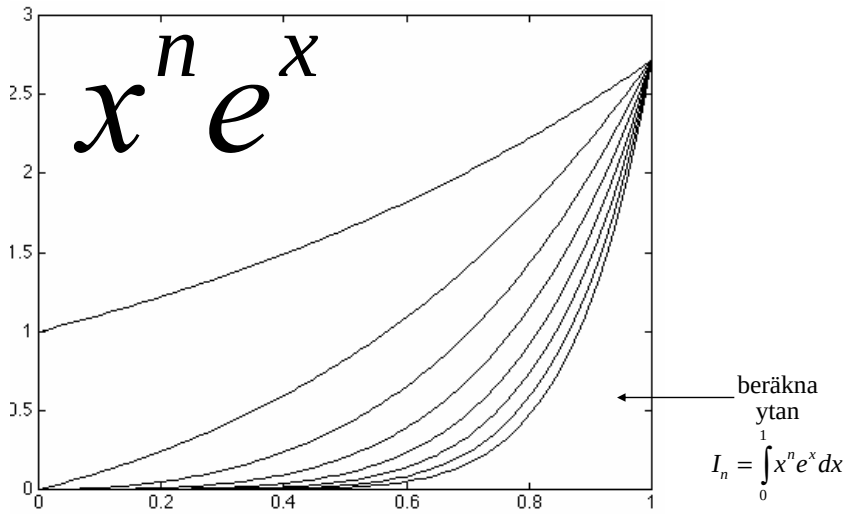
Ett exempel

- Jordens yta beräknas med:

$$A = 4 \pi r^2$$

- Approximationer:
  - modelleras som en sfär
  - radiens värde är resultat av mätningar och tidigare beräkningar
  - värdet för π är ett oändligt värde som måste trunkeras
  - värden och resultat i beräkningarna avrundas
  - ... och dessutom ökar felets storlek p.g.a beräkningarna (bl.a. multipliceras felen med 4)

Exempel på instabil beräkning



$x^n e^x$ ...

- Kan beräknas m.h.a. rekursionsformeln

  $I_{n+1} = e - (n+1) \cdot I_n$

- I_0 ges av

  $I_0 = e - 1 = 1.71828\ldots$

- Kan sättas in och ger

  $I_1 = e - I_0 = e - (e - 1) = 1$

  $I_2 = e - 2 \cdot I_1 = 2.71828\ldots - 2 \cdot 1$

  $= 0.71828\ldots$

  $I_3 = e - 3 \cdot I_2 = \ldots$

$x^n e^x$ ... beräkning i Java

```
I(1)  =      1.0
I(2)  =      0.7182818284590451
I(3)  =      0.5634363430819098
I(4)  =      0.4645364561314058
I(5)  =      0.395599547802016
…
I(15) =      0.1604958541638526
I(16) =      0.15034816183740363
I(17) =      0.16236307722318344
I(18) =     -0.2042535615582568
I(19) =      6.599099498065924
I(20) =   -129.26370813285942
I(21) =   2717.256152618507
I(22) = -59776.91707577869
…
I(98) = -5.013444424129383*E137
I(99) =  4.963309979888089*E139
I(100) = -4.963309979888089*E141
```

Rätt hit!

Matematiskt ...

då $0 < x < 1$ gäller

$x^n > x^{n+1}$ för alla $n >= 0$

därför måste gälla att

$I_0 > I_1 > I_2 > \ldots > I_{100}$

och alla $I_n > 0$

- Men varför blir det då fel???

Fel introduceras

- Relativa noggrannheten i varje beräkning är 10^{-16}

$I_2 = 0.71828\ldots \pm 10^{-16}$

$I_3 = e - 3 \cdot I_2$

$= e - 3 \cdot (0.71828\ldots \pm 10^{-16}) \pm 10^{-16}$

$= 0.5634\ldots \pm 4 \cdot 10^{-16}$

- I varje iteration/steg
  - tillfogas ny avrundning
  - multipliceras tidigare fel med 3

➔ Felet blir större och större

➔ Storleken av felet öka allt fortare

x^n e^x ... igen

- Kan beräknas m h a rekursionsformeln

$$I_{n+1} = e - (n+1) \cdot I_n$$

- Om man istället räknar "baklänges"

$$I_n = \frac{e - I_{n+1}}{n + 1}$$

- ... är 10^{-16} övre gräns för avrundningsfelet

$$(\text{fel i } I_n) = \frac{1}{n+1} (\text{fel i } I_{n+1}) \pm 10^{-16}$$

- Denna beräkning sägs vara ***stabil***, dvs "de gamla avrundningsfelen växer inte"

Rundningsregler

- Avkorta
- Avrundning till närmaste jämna

- Vancouver-exemplet:
  - jan -82 börs-index satt till 1000
  - nov -83 bottennotering 520

- Index noterades med tre decimaler och avhuggning
- Räkningar med avrundning nästan dubblade index

Kancellation och andra problem

- Noggrannhetsförlust vid subtraktion
  - två nästan lika stora tal
  - ➔informationen försvinner i den fortsatta beräkningen
- Stora tal kan slå ihjäl små tal
  - ➔antal signifikanta tecken räcker ej
  - ➔Se `NumericTest.java`
- Interna talrepresentationen påverka noggrannheten
  - ➔`0.1` i binär representation blir inte noggrann
  - ➔Se `LoopTest.java`

➔Lösningar:
  - ➔matematiskt ekvivalenta omskrivning av uttrycken
  - ➔användning av klasserna `BigInteger` och `BigDecimal`

Innehåll

- OOP snabbintroduktion
- Programvaruutveckling och programmering
- Datatyper
- Uttryck
- Satser
- Arv, polymorfi och dynamisk bindning
- Att organisera Javakod
- Metodik och klassdesign
- Design (CRC)
- Fält
- Undantag
- In-/utmatning och filer
- Grafik
- Rekursion
- GUI:s
- Applets vs applikationer
- Interfaces och sortering
- Noggrannheten i beräkningar
- Kort om filer

Filer

- Java I/O baseras på ***streams*** (se F9)

stream	syfte	enhet	datatyp
`System.in`	läsa input	tangentbord	`InputStream`
`System.out`	skriva output	bildskärm	`PrintStream`
`System.err`	skriva fel	bildskärm	`PrintStream`

- `Java.io` paketet innehåller klasserna `FileReader` och `FileWriter` för hantering av textfiler
- De måste kopplas ihop med `BufferedReader` och `BufferedWriter` på motsvarande sätt som vid vanlig I/O

Textfiler

- Filnamnen anges vid instansiering, som
  - Sträng, eller
  - `File` objekt, eller
  - `FileDescriptor` objekt
- `Swing` tillhandahåller en `JFileChooser` komponent för att interaktivt välja en fil

- Se `Inventory.java` (läsa från fil)
- Se `TestData.java` (skriva på fil)

Att skriva objekt

- Objekt kan skrivas genom ***serialisering***
- Enkel serialisering åstadkommas med `ObjectOutputStreams writeObject` metod

- Skriver ut värden på alla attribut (även de ärvda och privata)
- Anropas rekursivt om ett värde är en referens

- Motsvarande deserialisering görs med `ObjectInputStreams readObject` metod

Slut och tack