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
- GUI:s
- Applets vs applikationer
- Rekursion
- Interfaces och sortering
- Noggrannheten i beräkningar

Felhantering

- Vissa fel kan inte förebyggas med hjälp av if-satser
  - En fil kan t ex raderas under skrivningen/läsningen, fast man har kollat i förväg att allt är OK
- Möjliga åtgärder:
  - Avbryt eller avsluta programmet
  - Ignorera felet och fortsätt
  - Skriv ut ett meddelande och fortsätt
  - Ignorera anropet som ledde till felet
  - Tolka anropet på ett meningsfullt sätt
  - Kräver åtgärd från användaren interaktivt
  - Producera ett felaktigt resultat
  - Returnera en felkod
  - ...

Vad är bäst?

- Många alternativ
- Ingen åtgärd bäst i alla lägen
- Beror på anroparens mål
- ➨ Låt anroparen bestämmer

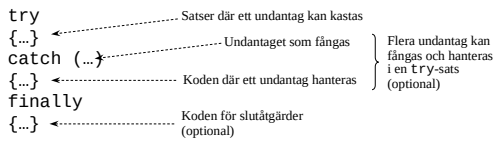
- ➨ Throw/ raise exception
- ➨ "Kasta"/ "flagga" ett undantag

- ➨ Överlåta åt anroparen att genomföra lämplig åtgärd

Undantag

- En ***exception*** är ett objekt som signalerar ett undantag

- `throw`-satsen kastar ett undantag (se ex. i fig 8.8 s.526)
- … som kan fångas m h a en `try`-sats



Fånga exceptions

```
try
{
  int a = b / c;
}
catch (DivByZeroException e)
{
  a = 9999;
}
```


Två kategorier av undantag 1

- ***Checked exceptions***
  - Fel som vanligtvis inte beror på fel i logiken av programmet
  - Fel som kan uppträda under "normala" förhållanden
  - Måste fångas (dvs kod som kan kasta en checked exception måste skrivas i en `try`-sats) eller kastas vidare med metoden, t ex
  - ➨ `IOException`

- ***Unchecked exceptions***
  - Fel som brukar beror på fel i logiken av programmet
  - Fel som kan uppträda under "normala" förhållanden
  - Får ignoreras (men det är god programmeringsstil att fånga eller deklarera det), t ex
  - ➨ `ArrayIndexOutOfBoundsException`

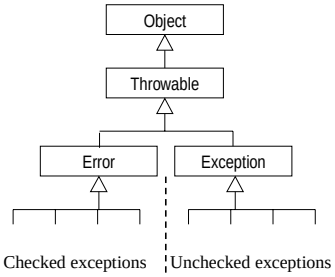
Deklarera ett undantag

- Exempel:

```
public static void arraycopy (
   Object source, int srcindex,
   Object dest, int destindex, int size)
throws ArrayIndexOutOfBoundsException,
       ArrayStoreException
```

Försökt spara fel sorts objekt

Två kategorier av undantag 2



Mera undantag

- Om ett undantag inte fångas ***propageras*** det vidare till anroparen
- Programmeraren kan definiera egna undantag genom att ärva från en `Throwable` subklass
- Checked exceptions får bara kastas i en `throw` block

- Se t.ex. Figure 8.8

Innehåll

Input/ output

- Java I/O baseras på ***streams*** (strömmar)
- En ström är en ***sekvens av bytes*** som flödar från en källa till ett mål
- Fördefinierade ***standardströmmar***:

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

- Standard strömmarna kan läggas om med `System.setIn/Out/Err` metoderna

`PrintStream`

- Huvudsakligen `print` och `println` metoder som skriver på standardoutput
- `Print` och `println` är överlagrade för alla primitiva datatyper, `String` och `Object`

➨ `Print` och `println` finns för alla tänkbara datatyper

InputStream

- Input strömmar är mera komplicerade
- `InputStream` är en abstrakt klass
- `System.in` måste därför "mappas" till en specifik ström med de egenskaper man vill ha
  - Läsa in strängar
  - Läsa in heltal
  - Läsa in …
- Man behöver `BufferedReader` och `StringTokenizer`
- För att underlätta input definieras oftast hjälpklasser
- Se Thomas J:s `In` klass

static

- Attribut och metoder kan deklareras `static`
- Associerar attribut och metoder med klassen snarare än objektet
- Avsteg ifrån objekts-tanken

Statiska attribut

- Normalt har varje objekt eget minnesutrymme
- `Static` gör variabeln gemensam för alla objekt i klassen (dvs den delas av alla)

```
private static int count;
```

- Kallas ibland ***klass variabler***

Statiska metoder 1

- Statiska metoder anropas via klassnamnet
- Man behöver inte skapa ett objekt först
- Därför kallas de också ***klass metoder***
- Exempel: Klassen `Math` i paketet `java.lang` innehåller statiska matematiska operationer

```
Math.abs (num)  -- absolutbeloppet
Math.sqrt (num) -- kvadratrot
Math.random ()  -- slumptal i intervallet [0.0…1.0[
```


Statiska metoder 2

- Statiska metoder får inte referera till instansvariabler (eftersom de instansieras ju först när ett objekt skapas)
- ➨ Får bara referera statiska variabler eller lokala variabler (som existerar oberoende av objekt)
- Även metoderna är specifika för objekt
- ➨ Får bara anropa statiska metoder

Klassen `In` 1

```
import java.io.*;
import java.util.StringTokenizer;

public class In
{
  static StringTokenizer reader;
  static BufferedReader stdin;
  In()
  {
    reader = null;
  }

  public static String readLine() throws IOException
  {
    BufferedReader stdin = new BufferedReader(new
   InputStreamReader(System.in));

    return stdin.readLine();
}// readLine
```

```
public static String readWord() throws IOException
  {
    if (reader == null) reader = new StringTokenizer(readLine());
    if (reader.countTokens() == 0)
    {
      while (reader.countTokens() == 0)
      {
        reader = new StringTokenizer(readLine());
      }
    }
    return reader.nextToken();
  }// readWord

  public static int readInt() throws IOException
  {
    return Integer.parseInt(readWord());
  }// readInt

  public static float readFloat() throws IOException,
   NumberFormatException
  {
    return Float.valueOf(readWord()).floatValue();
  }
}
```

2

Klassen `String`

- Finns i paketet `java.lang` (importeras automatiskt)
- Motsvarar char[]
- Strängar är ***oföränderliga***
  - ➡ Effektiv
  - ➡ "Ofarligt" (inga alias)

Se API beskrivningen och App.C s.719

- Många operationer och operatorer

  `"enSträng" + "en till Sträng"`

- Många bibliotekslasser som hanterar strängar

  `StringBuffer, StringTokenizer, …`

Wrapper-klasser

- Finns i paketet `java.lang` (importeras automatiskt)
- Gör objekt av primitiva typer

- Många metoder (se 5.8 och App.C s.709 ...)

```
int heltal = Integer.parseInt("123");
```