TDDD78
Objektorientering: Namnrymder

Hur *organiserar* man klasser i paket / namnrymder?

jonas.kvarnstrom@liu.se – 2017

Namnrymder

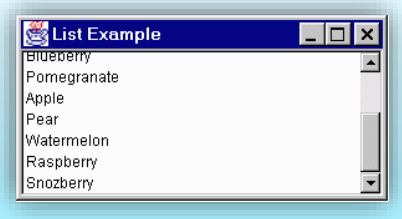
jonkv@ida

- **Enkla klassnamn** kan ge **namnkollisioner**
  - Datastrukturen List och GUI-elementet List?
    - Kan döpa om to ListStructure, ListComponent...
    - ➔ Måste alltid använda långa namn, även om vi inte jobbar med GUI
  - Din liststruktur och någon annans?
    - Döp om till JonasListStructure, …?

Många språk har distinkta namnrymder

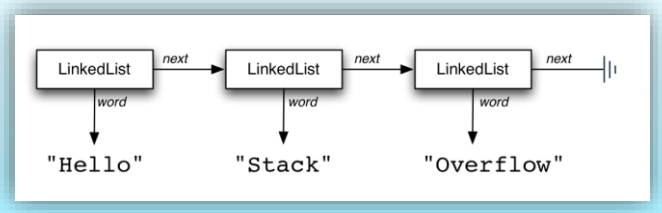
"GUI-namnrymd"

List ➔



"Datatyps-namnrymd"

List ➔



Namnrymder ger oss enklare namngivning, men också organisation

Repetition: Namnrymder i Python

jonkv@ida

- Från Python:

```
>>> import newton
This program tries to find the square root of a number.
Enter a number: 2
1.5
1.4166666666666665
1.4142156862745097
1.4142135623746899
1.414213562373095
>>> newton.find_root()
This program tries to find the square root of a number.
Enter a number: 10
3.5
3.178571428571429
3.162319422150883
3.162277660441363
3.16227766016837
```

Gör "newton" till en namnrymd (med innehållet från filen newton.py)

Fullständigt namn: namnrymd.funktion

Ger möjlighet till t.ex. "gui.List" och "datatypes.List"

Paket 1: Namnrymder

jonkv@ida

- I Java ingår **varje klass** i ett **paket** (som är en namnrymd)
  - ```
    package se.liu.ida.jonkv.graphics.shapes;
    class Circle {
        ...
    }
    ```
  - ```
    se.liu.ida.jonkv.graphics.shapes.Circle c1;
    ```

Standard för **unika namn**:
Börja med domännamn, baklänges

Python	Java
Om vi gör "import" skapar vi en namnrymd, där filens innehåll placeras	**Klassen deklarerar vilken namnrymd den tillhör – detta gäller alltid!**
Kod i fil A hittar bara kod i fil B om den "importeras"	**Skriv paketnamn.klass så hittas klassen utan "import"**

Hur hittar Java klassen?

Paket 2: Katalogstruktur

- Javas källkod lagras i en **katalogstruktur** motsvarande paketen
  - rotkatalog ~/mysource
    - katalog se
      - katalog liu
        - katalog ida
          - katalog jonkv
            - katalog graphics
              - katalog shapes
                - fil Circle.java
                - fil GraphicCircle.java

Kan bli många nivåer

Utvecklingsmiljöer hjälper till att hålla reda på detta…

- Ange rotkatalogen ~/mysource – så hittas allt annat från den
  - På kommandoraden: Miljövariabeln **CLASSPATH**
  - I övrigt: Beror på utvecklingsmiljön

Paket 3: Existerande

- Många paket finns i Javas klassbibliotek

java.lang:
Object, Class, String, Thread, …

java.util:
List, ArrayList, Date, Calendar, Timer, Formatter, …

java.io:
Reader, InputStream, …

Paket 4: Import förkortar namnen

- Men nu blir det väl lite jobbigt…
  - javax.swing.filechooser.FileFilter filter =
    new javax.swing.filechooser.FileFilter();

- Undvik med **Javas** variant av **import**
  - **package** se.liu.ida.jonkv.io;
    import javax.swing.filechooser.FileFilter;

```
public class FileSaver {
    …
    FileFilter filter = new FileFilter();
    …
}
```

Talar bara om var FileFilter finns!

Om jag *säger* FileFilter
menar jag
javax.swing.filechooser.FileFilter

Ungefärlig motsvarighet i Python:

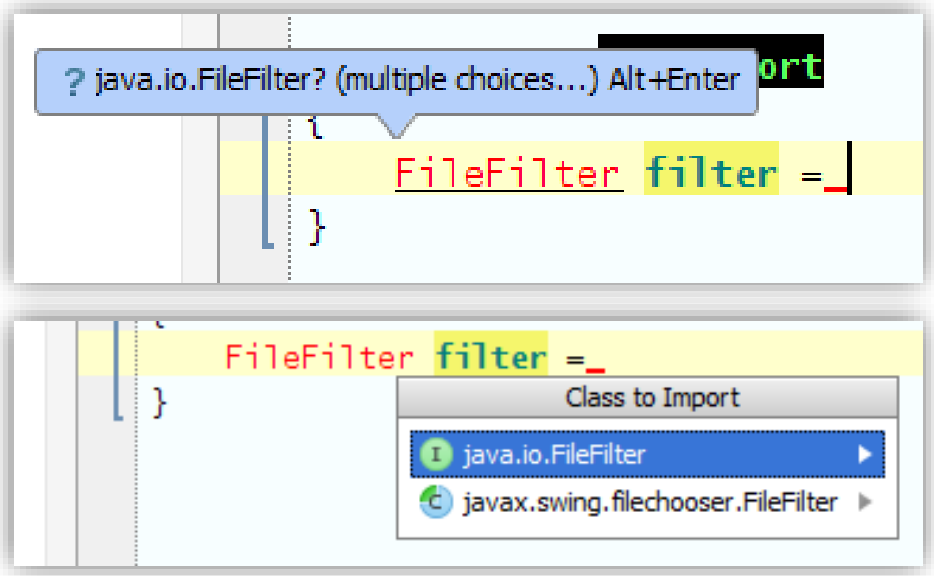
from javax.swing.filechooser
import FileFilter

Paket 5: Automatisk import

jonkv@ida

- Vissa klasser **importeras automatiskt**
  - Alla klasser i samma paket
  - Alla klasser i java.lang (”systemklasser” som Object, String)

- Moderna utvecklingsmiljöer hjälper till med annan import



Paket 6: Import med "*"

jonkv@ida

- Kan importera **alla klasser i ett paket** med "*"
  - `import javax.swing.filechooser.*;`
    `FileFilter filter = new FileFilter();`

Motsvarighet i Python:

```
from javax.swing.filechooser
import *
```

Undvik: Kan ge problem om nya klasser adderas till ett importerat paket…

Skriv i Java 1.0…

```
import java.awt.*;// Har en List-klass
import java.util.*; // Ingen List-klass…
class MyClass {
   List list; // Unikt!
}
```

Kompilera med Java 7…

```
import java.awt.*;// Har en List-klass
import java.util.*; // Har också en…
class MyClass {
   List list; // ???  Kompileringsfel!
}
```

Paket 7: Använd dem!

jonkv@ida

- Om inget paket anges:
  - Klassen hamnar i "unnamed package"
  - **Använd inte detta** – ange alltid paketnamn!

- För att:
  - Undvika namnkollisioner
  - Förbättra läsbarhet, underhåll, organisation:
    Vilka klasser är **relaterade**?
  - Undvika problem med verktyg som antar att paketnamn finns

De flesta av projekten i kursen
bör ha flera "delpaket"