ex1enumwrapper

Joachim von Hacht

Uppräkningstyper

```
enum WeekDay {   // Enumeration type
   MON, TUE, WED, THU, FRI, SAT, SUN
}

WeekDay d1 = WeekDay.FRI;
WeekDay d2 = WeekDay.THU;


// Loop through all days
for (int i = 0; i <  WeekDay.values(); i++) {
   // Do something
}
```

2

Ibland behövs ett begränsat antal värden av en viss typ (typen innehåller ett litet antal värden)

- T.ex. veckodagar, färger , etc
- VI skulle kunna använda String för dessa t.ex. “Mon”, “Tue” (eller integer med Månd = 1, o.s.v.) men …
  - … detta blir inte typsäkert …
  - t.ex. om vi stavar fel, t.ex. “Tui”, så släpper kompilatorn igenom felet (felet kommer senare under körning)
  - Om vi använder int för dagar så kan någon av misstag tilldela en dag värdet -12, o.s.v. …
- Bättre att låta typsystemet se till att allt stämmer

Genom att deklarera en **uppräkningstyp** (enumeration**)** skapar vi en ny (egen) klasstyp med ett antal (uppräknade) värden (en enum är en slags klass).

- Vi kan därmed deklarera variabler av denna typ.
- Kompilatorn kan kontrollera att vi bara använder korrekta värden !

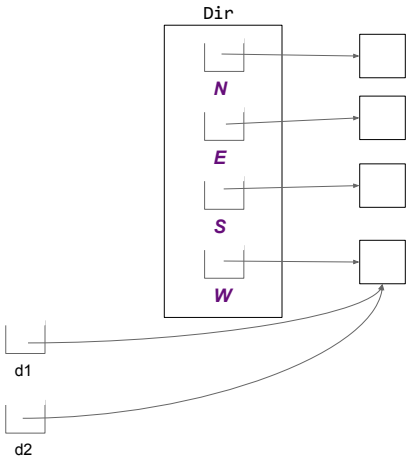
Deklarationen av uppräkningstyp görs men det reserverade order **enum**.

- De värden som tillhör typen räknas upp (vi skriver namnen på värdena)
  - I bilden: MON, TUE, … o.s.v. (stora bokstäver skall användas)

- Värdena är av typen WeekDay (INTE String, inga citattecken runt)
- För att komma åt värdena måste vi använda **punktnotation** d.v.s. typnamn.värde

Likhet

```
enum Dir {
   N, E, S, W
}
```




```
Dir d1 = Dir.W;
Dir d2 = Dir.W;

if( d1 == d2 ){
 // True
}
```

3

Det finns exakt ett (icke-ändringsbart) objekt för varje värde i enum:en vi räknar upp.

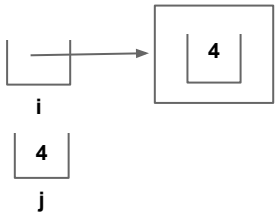
- Likhet (==) fungerar därför som mellan värden (eftersom det är identitet i detta fall)

Omslagstyper

```
// Wrapper type Integer
Integer i = 4;    // Boxing

int j = i;        // Unboxing

Double d1  = 5.32;   // Boxing
Character ch  = 'X';
```



4

Omslagstyper(wrapper types) är <u>referenstypsversioner av primitiva typer</u>

- Finns färdiga i Java
- De paketerar in ett primitivt värde i ett objekt, t.ex. int packas in i en Integer-objekt
- Omvandling mellan omslagstyp och primitiv typ sköts automatiskt **(boxing/unboxing)**
  - Kan (un)boxa för hand men behövs sällan
- Objekt av omslagtyper kan inte ändras (icke muterbara) men det "verkar" som om de kan ändras eftersom boxing/unboxing kickar in.

Det finns omslagstyper för alla primitiva typer, …

- T.ex. Boolean och Character.

Vi behöver omslagstyper då vi använder generiska metoder, mer strax.

Omslagstyper, Boxing och Likhet

```
Double d1 = 5.0;
Double d2 = 5.0;
out.println(d1 == 5.0);       // True, unboxing

out.println(d1 == d2);        // False! No boxing
out.println(d1.equals(d2));   // True, better use!

out.println(d1 == 0 + d2);    // True, unboxing
out.println(d1 >= d2);        // True, unboxing

Integer i1 = 99;
Integer i2 = 99;
out.println(i1 == i2);       // True! ... caching!
```


Omslagstyper håller referenser till objekt.

- Om jämförelsen görs mot en primitiv typ sker unboxing och värdena jämförs.
- Jämförelser förutom likhet unboxas
- För värdelikhet måste man annars använda equals().
  - Förutom för små heltal (< 128) där finns det färdiga objekt, d.v.s. == fungerar
  - Kallas : caching (av effektivitetsskäl)