DD2385

Programutvecklingsteknik

Några bilder till föreläsning 4

7/4 2014

Innehåll

- Abstrakta klasser
- Klasshierarki och typhierarki
- Polymorfism och dynamisk bindning
- Polymorfi-exempel: Schack
- UML-översikt
- Klassen Object i Java

Abstrakt klass

- Inleds med `abstract class`
- Kombination av vanlig klass och interface
- Kan innehålla vanliga metoder och variabler
- Innehåller oftast minst en abstrakt metod som definieras i subklass
- Objekt av klassen kan *inte* skapas

Exempel: Implementation av Schackpjäser

- De olika pjäserna har mycket gemensamt
- Men förflyttning sker på olika sätt
- Definiera en abstrakt klass med allt gemensamt
- Definiera en konkret subklass för varje pjästyp:
  - Bonde
  - Springare
  - Löpare
  - Torn
  - Dam
  - Kung

Exempel: Skiss av klasser

```
abstract class Schackpjas {
  Color farg; Image bild;
  Brade brade; Ruta ruta;

  Schackpjas (...) {...}   // konstruktor

  abstract boolean dragOK(Ruta r1, Ruta r2);
}
```

```
class Bonde extends Schackpjas {
  boolean dragOK(Ruta r1, Ruta r2) {
    // implementera bondens drag
  }
}
```

Konstruktor m.m. visas ej.

Exempel: Skiss av klasser

```
class Dam extends Schackpjas {
  boolean dragOK(Ruta r1, Ruta r2) {
    // implementera damens drag
  }
}
```

```
class Torn extends Schackpjas {
  boolean dragOK(Ruta r1, Ruta r2) {
    // implementera tornets drag
  }
}
```

```
class Springare extends Schackpjas {
  boolean dragOK(Ruta r1, Ruta r2) {
    // implementera springarens drag
  }
}
```

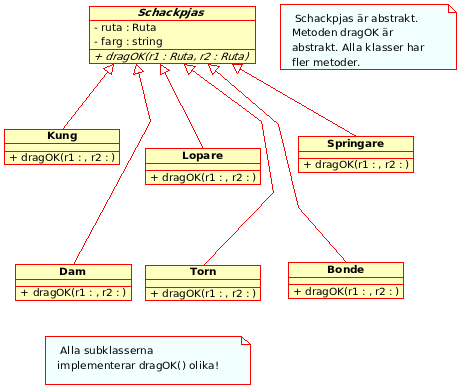
Klasshierarki = Typhierarki

- Klasshierarkin är en trädstruktur
- Referenser av viss typ får referera till objekt av typer som är nedanför i trädet.
- Schackpjas får referera till Bonde, Dam, Kung, Torn ...
- En Schackpjas har många "former" (Bonde, Dam ...): POLYMORFISM

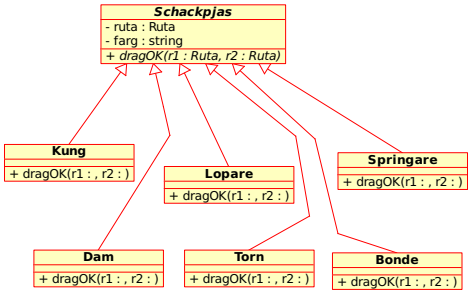
Dynamisk bindning

- En Schackpjas-referens "ser" endast Schackpjas-definitioner, t.ex. `dragOK`
- Om en metod, t.ex. `dragOK`, definierats om i en subklass så används den omdefinierade!
- Objektetets typ och inte referensens typ "väljer" metod. Detta är dynamisk bindning
-
```
Schackpjas pjas = new ...//Dam, Kung, Torn...
...
pjas.dragOK(..)  //Dynamiskt metodval
                 //beroende av typen
                 //hos pjas-objektet
```

Klass/typ - hierarki för Schackpjäserna
med UML-notislappar

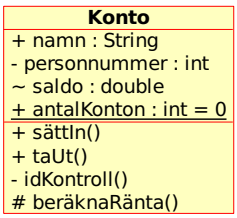


Klass/typ - hierarki för Schackpjäserna



`dragOK()` är en *abstrakt* metod.
`dragOK()` implementeras olika i `Kung, Dam, Lopare ....`
Alla klasserna har fler metoder

UML – vanlig klass



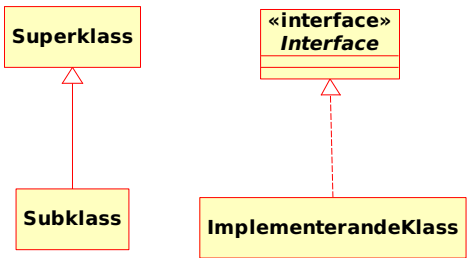
Variabler, ev. med typ

Metoder,
ev. typ,
ev. parameterlista

~ *paketsynlighet*
`# protected`
`+ public`
`- private`
`understruket` betyder `static`

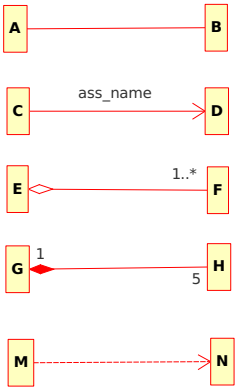
Java-specifik UML

UML – arv och implementation



Relationen ÄR

UML - associationer



Oriktad eller dubbelriktad association

Riktad association,
har, känner till

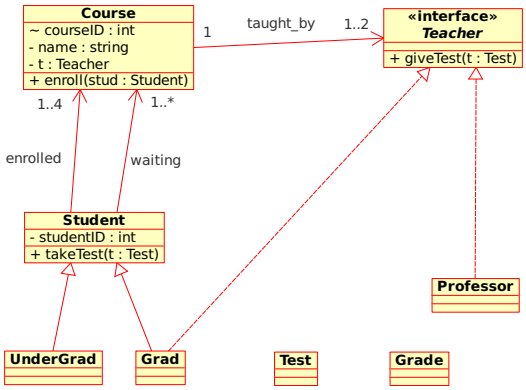
Aggregat, har, består av,
äger ej

Komposition
har, består av, **äger**

Beroende
beror av,
metodparameter, lokal variabel

UML - dubbelriktad association

UML - exempel



Klassen Object

- Superklass till *alla* Javaklasser
  - Överst i klasshierarkin
  - Mest generella typen i Java
  - `Object` - referens kan referera till alla objekt
  - Innehåller 11 metoder
- *Alla* klasser har 11 metoder från `Object`
- Flera metoder ärvs tomma eller väldigt enkla, kan/bör definieras om av subklasserna.

Metoderna i Object

String toString() String-repr.
boolean equals(Object o) likhet?
int hashCode()
Object clone() kopia
void finalize() anropas före GC
Class getClass() meta-information
void notify() väck *en* tråd
void notifyAll() väck *alla* trådar
void wait() lägg tråd
void wait(t) i
void wait(t,n) vänteläge

Jämförelse med == eller `equals()` ?

`==`
jämför variablers innehåll
primitiva typer: enkelt och självklart
referenstyper: samma princip

`equals()`
kan jämföra objekts innehåll
om man definierat om den

```
Spelkort s1 = new Spelkort("RUTER",5);
Spelkort s2 = new Spelkort("RUTER",5);
Spelkort s3 = s2;
```

`s1 == s2` ger **false** inte samma objekt men lika
`s2 == s3` ger **true** samma objekt

`s1.equals(s2)` kan ge **true**
om man definierat den rätt