Objektorienterad Programmering (TDDC77)

Föreläsning X: Klass diagram, inkapsling, arv

Ahmed Rezine

IDA, Linköpings Universitet

Hösttermin 2017

Outline

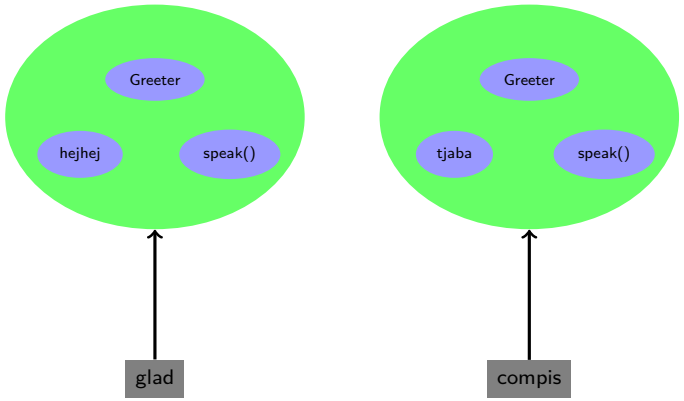
Att instansiera en klass

- Man instansierar (skapar ett objekt av) en klass genom att anropa en konstruktor
- Detta sker genom att använda nyckelordet **new** följt av klassnamn och parameterlista
- Det kan finnas flera konstruktorer med olika parameterlistor
- Exakt en instans skapas för varje gång man kör konstruktorn

```
Greeter glad = new Greeter("hejhej");

Greeter compis = new Greeter("tjaba");
```


Objekt

```
Greeter glad = new Greeter("hejhej");

Greeter compis = new Greeter("tjaba");
```



Konstruktorer

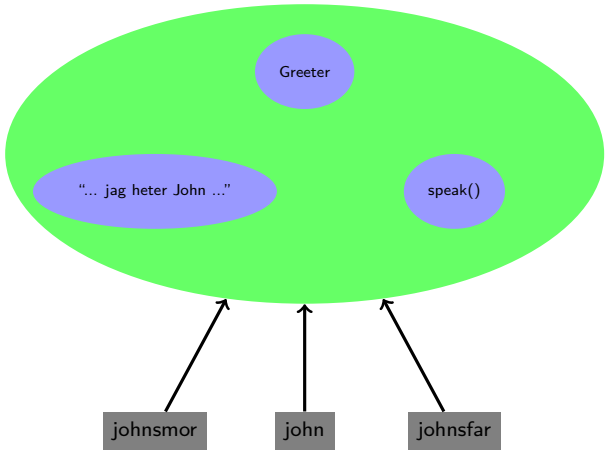
- En konstruktor saknar returtyp och heter samma som klassen
- Om ingen konstruktor skrivs så skapas automatiskt en så kallad standardkonstruktor som inte tar några argument
- Om man skapar en egen konstruktor så skapas inte standardkonstruktorn automatiskt
- Man kan ha godtyckligt många konstruktorer så länge de har olika parameterlistor
- Detta gäller generellt för alla funktioner och kallas överlagring (tas upp senare)

Referenser

- Objekt är referenstyper,det betyder att när man skapar ett objekt egentligen man får en referens tillbaka
- Gör man sedan en tilldelning eller jämförelse så är det referensen som jämförs

```
Greeter john = new Greeter("God morgon, jag heter John Smith!");
Greeter johnsmor = john;
Greeter johnsfar = johnsmor;
```


Flera referenser - samma objekt



Outline

Åtkomst

- I Java finns fyra åtkomstnivåer för variabler och funktioner
- **public** : alltid synlig
- **protected**: tas upp senare
- **default (package)**: tas upp senare
- **private**: endast synlig inom klassen
- i den här kursen håller vi oss till **public** och **private**

Åtkomst grundregel

- Deklarerar alla variabler som private
- Deklarerar alla funktioner som public
- Detta bettyder att om man vill kunna läsa/skriva en variabel från en annan klass så måste man göra funktioner som utför det. Detta kallas för getters och setters och eclipse kan generera den automatiskt
- Det finns undantag, men dessa kommer ni att lära er att känna igen så småningom
- Konstruktorer bör vara deklarerade som public

Outline

Abstrakt datatyp

- Ibland vill man behandla data på ett sätt som inte riktigt stämmer med de primitiva datatyperna.
- Man måste då “kombinera ihop” en ny datatyp som är anpassad för det man faktiskt vill göra med den
- Ett exempel på detta är stacken som ni implementerade till lincalc
- En abstrakt datatyp är ofta en kombination av variabler (tillstånd) och funktioner (tjänster) ...
- ... det låter som att abstrakta datatyper och klasser har en del gemensamt

Öva - gör om stacken som en klass

- Skapa en klass StringStack som kan lagra 100 strängar
- Strängarna ska lagras i en Array som det inte ska gå att ändra i så att stacken hamnar i ett otillåtet tillstånd
- Ska klara operationerna **push**, **pop** och **isEmpty**

StringStack

```
public class StringStack{
    private static int MAX = 100;
    private String[] data = new String[MAX];
    private int pos = 0;

    public boolean isEmpty(){
     return pos == 0;
    }

    public void push (String s){
     data[pos++] = s;
    }

    public String pop(){
     return data[--pos];
    }
}
```


Outline

UML

- UML = Unified Modeling Language
- En enorm standard för att modellera precis allt
- En liten del kallas för klass-diagram och beskriver hur klasser hänger ihop med varandra
- Vi kommer i kursen att använda en mycket begränsad delmängd av dessa klass-diagram kallad Lättviktsdiagram

Lättviktsdiagram

- De som ska rätta era uppgifter behöver förstå era design och vill därför ha diagram som beskriver era klasser och relationerna dem emmellan
- Ni behöver inte göra formellt korrekta UML-diagram

Klassen calculator

Calculator
result: double
Calculcator() Calculcator(double) add(a: double): double add(a: double, b: double): double reset():void

```
class Calculator{
  private double result;

  public Calculator(){
    result=0;
  }

  public double add(double a){
    result += a;
    return result;
  }

  public void reset(){
    result = 0;
  }
}
```


Klasser

- Klasser ritas som tredelade rutor
- Översta rutan innehåller klassnamnet
- Därefeter följer attribut (alltså variabler)
- Understa rutan innehåller metoderna

KlassNamn
attribut1Namn : attribut1Typ attribut2Namn : attribut2Typ ...
metod1Namn(param1Namn: param1Typ, ...) : returTyp1 ...

Outline

Klassen calculator

Calculator
memory: double result: double
Calculator() Calculator(mem: double) add(a: double): double add(a: double, b: double): double store():void load():void

```
class Calculator{
  private double result;

  public Calculator(){
  result = 0;
  }

  public Calculator(double mem){
  result = mem;
  }

  public double add(double a){
    result += a;
    return result;
  }

  public double add(double a, double b){
    result = a + b;
    return result;
  }

  public void reset(){
    result = 0;
  }
}
```


Signatur

```
public static int parseInt(String s)  throws NumberFormatException
```

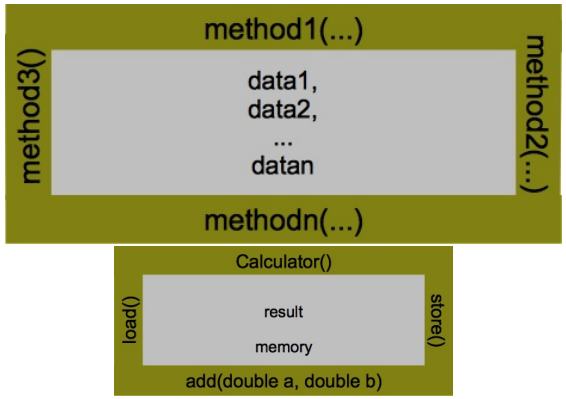
- En konkret metod deklareras med hjälp av sex komponenter
  - Accessmodifierare såsom public och protected
  - Returtyp eller void
  - Metodens namn (identifierare) som ska börja med liten bokstav
  - Parameterlista med noll eller fler datatyper och identifierare
  - Eventuella “checked exceptions” som metoden kastar
  - Metodens kropp inom måsvingar
- Metodens signatur betäms av metodens namns samt datatyper och ordning på parameterlistan

Överlagring

- En metod överlagrar en annan om den har samma namn (identifierare) men olika signatur
- Returtyp spelar ingen roll för överlagring vilket innebär att två metoder med olika returtyp men samma signatur inte är tillåtet (kompilatorn kan inte skilja på dem).

Outline

Inkapsling (encapsulation)



Inkapsling (encapsulation)

	public	private
variabler	bryter mot inkapsling	verkställer inkapsling
metoder	tjänster till andra klasser	hjälpar egna metoder

Personnummer

```
class Personnummer{
  private int year, month, day, birthNumber, controlNumber;

  public Personnummer(String str){
    year = Integer.parseInt(str.substring(0,4));
    month = Integer.parseInt(str.substring(4,6));
    ...
  }

  public void set_year(int _year){
    year = _year;
  }
  public void set_month(int _month){
    month = _month;
  }
  ...
  public int getYear(){
    return year;
  }
  public int getMonth(){
    return month;
  }
  ...
}
```

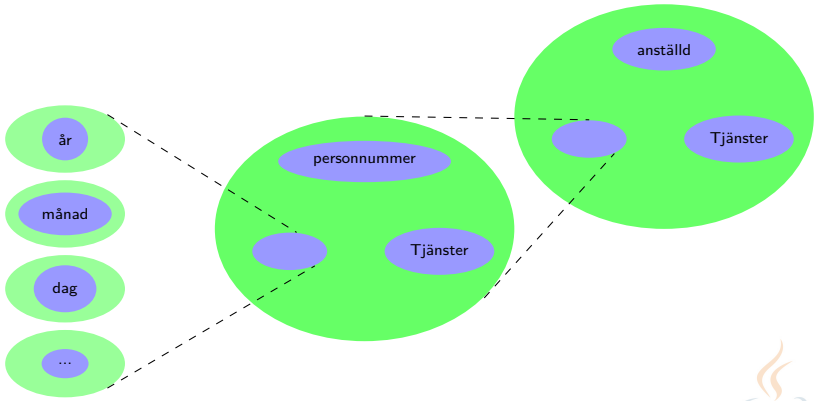
Employee

```
class Employee{
    private Personnummer personnummer;

    public Personnummer getPersonnummer(){
       return personnummer;
    }

    public void readData() {
        Scanner in = new Scanner(System.in);
        System.out.println("Personnummer?");
        personnummer= new Personnummer(in.nextLine());
    }
}
```

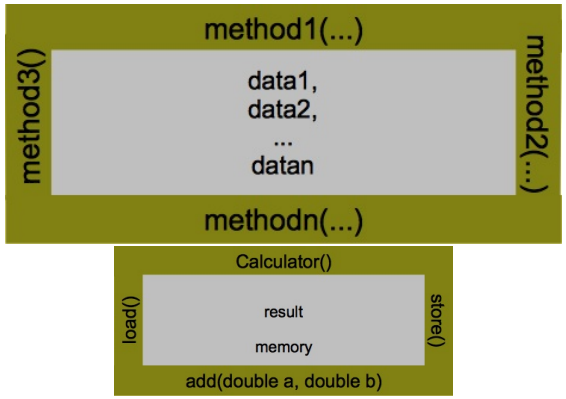
Driver

```
class Driver{
  public static void main(String[] args){
     Employee e = new Employee();
     e.readData();
     System.out.println("The employee was born on: "
                        + e.getPersonnummer().getYear() );
    }
}
```


Inkapsling (encapsulation)



Inkapsling (encapsulation)



Inkapsling (encapsulation)

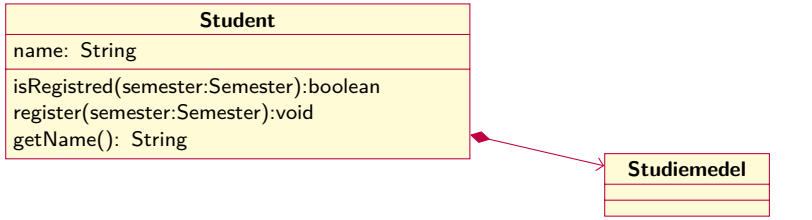
	public	private
variabler	bryter mot inkapsling	verkställer inkapsling
metoder	tjänster till andra klasser	hjälpar egna metoder

Inkapsling (encapsulation)

- Anta att klass A kapslar in klass B. Detta illustreras genom att ett streck ritas från A till B, samt att en ifylld romb ritas vid A, alltså den klass som kapslar in den andra.

Inkapsling (encapsulation)

- Student inkapslar Studiemedelw



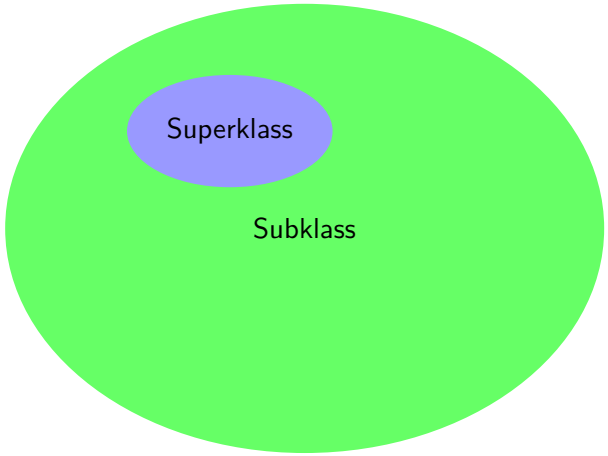
Outline

Arv (inheritance)

- Är en relation

```
class Student{
    private int points;
    private Studiemedel studiemedel;
}

class ITare extends Student {
    private int pointsDatavetenskap;
}
```

Arv (inheritance)



Protected eller getters och setters

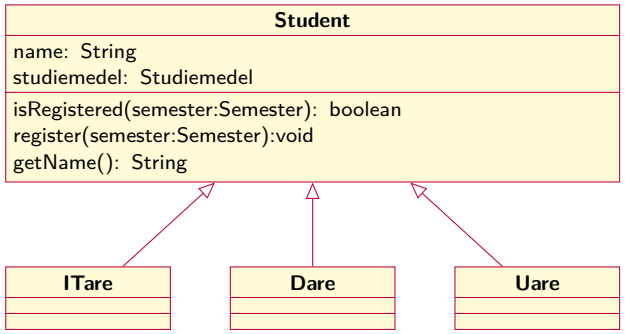
- Vill man komma åt instansvariabler i basklassen så fungerar inte det om de är deklarerade som private.
- Lösning: deklarera som protected : som private men synlig i subklasser
- Bättre lösning: använd getters och setters

Att prata om arv

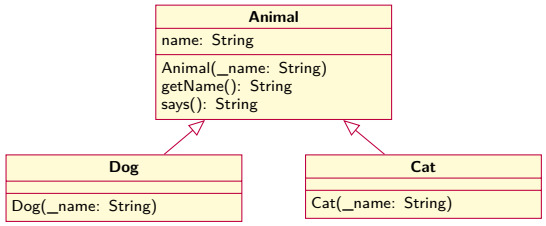
- Subklasser ärver tillstånd och tjänster från superklasser
- Ett sätt att dela på gemensam kod
- En subklass utöker eller specialiserar sin superklass
- En subklass är en subtyp till superklassens datatyp
- En superklass kan ha flera subklasser

Arv (inheritance)

- Antag att klass B ärver från klass A. Detta illustreras genom att en icke ifylld “pil” ritas från B till A



Polymorfism via Arv (inheritance)




```
Animal hunden = new Dog("bobby");
Animal katten = new Cat("kitty");

System.out.println(hunden.getName() + " says " + hunden.says());
System.out.println(katten.getName() + " says " + katten.says());
```