Objektorienterad Programmering (TDDC77)

Föreläsning IX: Klasser och Objekt, Instantiering

Ahmed Rezine

IDA, Linköpings Universitet

Höstermin 2016

Förkunskaper från tidigare föreläsningar:

- STONE
- Variabler, datatyper, typkonvertering
- Logiska och matematiska uttryck
- Metoder-returvärde och parametrar
- Villkorssatser if/else
- Loopar, for, while, do-while
- Arrayer, String

Vad händer i resten av kursen:

- Klasser och objekt (classes and objects)
- Inkapsling och arv (encapsulation and inheritance)
- Åsidosättning och överlagring (overriding and overloading)
- Abstrakta klasser och gränssnitt (abstract classes and interfaces)
- Samlingar (collections), generics
- Uppräkningar (enumerations)
- Undantag (exceptions)

Outline

Klasser och Objekt

Instansiering

Åtkomst

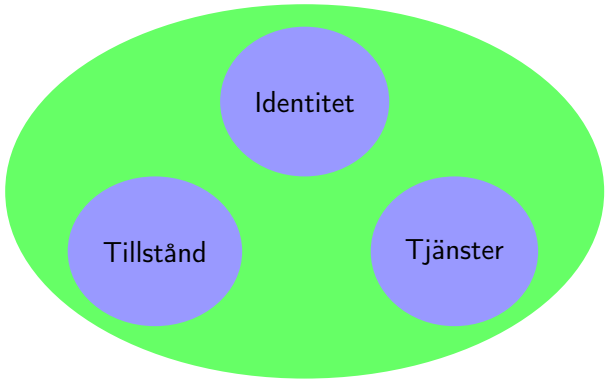
Calculator

Abstrakt datatyp

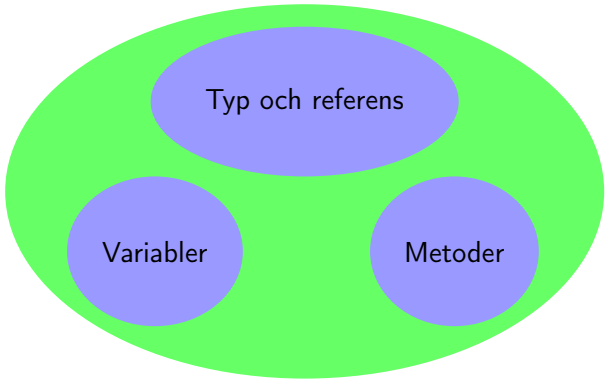
Objekt

- Identitet : Typ och referens
- Tillstånd: Variabler
- Tjänster: Metoder

Ett objekt



Ett objekt



Klasser av objekt

- Varje objekt som skapas måste höra till en klass
- Klassen definierar hur objekten fungerar, det vill säga deras beteende. Lite som en mall för objekten
- En klass skapas genom att man definierar den. Varje klass i en separat fil.
- Klassens namn blir objektens typ

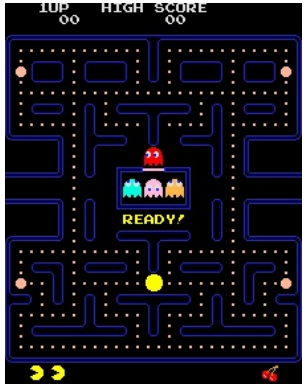
```
/* Greeter.java
 * Enkel Klass som representera en person som
 * hälsa med ett specifikt meddelande
 */
class Greeter{
    private String message;

    public Greeter(String msg){
     message = msg;
    }

    private String speak(){
     return message + "!!!";
    }
}
```

Objekt

- Ett objekt kan även kallas instans (förekomst) av en klass.
- När man skapar ett objekt som tillhör en viss klass så säger man att man instantierar klassen
- Varje objekt tar sina egna "kopior" av variablerna som definierades i klassen. Dessa bestämmer objektets tillstånd
- Ett objekt skapas med nyckelordet new

```
/* Driver.java
 * Deklarera och använda sig av Greeter programmet
 */
class Driver{

    public static void main(String[] args){
     Greeter person= new Greeter("hej");
     Greeter glad= new Greeter("hejhej");

     //   System.out.println("person Greeter says: "
     //            + person.message);

     System.out.println("glad Greeter says: "
                + glad.speak());
    }
}
```

Pac-man



Exampel

```
import java.util.Scanner;

 class Game {
    static Scanner scan = new Scanner(System.in);
    static Ghost greenGhost = new Ghost("green", 1, 4);
    static Ghost blueGhost = new Ghost("blue ", 0, 3);

    public static void main(String[] args) {
        String command;
        do{
         //ask the ghosts to update there coordinates
         greenGhost.update();
         blueGhost.update();
         //print the new coordinates
         System.out.println(greenGhost.print());
         System.out.println(blueGhost.print());
         //check if the user wants to quit
         System.out.print("\nWrite \"c\" if you want to continue: ");
         command=scan.next();
     }while(command.equalsIgnoreCase("c"));
    }
}
```

Exampel

```
 class Ghost {

    String name;
    int col=0,row=0;

    public Ghost(String name, int _col, int _row){
     this.name = name;
     col = _col;
     row = _row;
    }

    public void update(){
     col = (this.col==9? col-1: col+1);
     row = (row==9? row-1: row+1);
    }

    public String print(){
     return name + ": (" + row + ", " + col + ")";
    }
}
```

Static

- Tidigare har vi skrivit **static** lite överallt
- Från och med nu så kommer vi att använda oss mycket mindre av **static**
- Till exempel när vi skriver main-metoden

Två sorter variabler:

- För att förstå vad nyckelordet **static** innebär behöver man veta att det i själva verket finns ytterligare ett sätt att indela variabler i två kategorier som anknyter till objekt och klasser
- De två nya kategorierna kallas instans och klassvariabler
- Tidigare gjorde vi uppdelningen primitiva typer och referenstyper. Detta har ingenting med instans och klassvariabler att göra

Instansvariabler

- En instansvariabel är individuell för varje instans av en klass
- Det är denna sorts variabler ni i fortsättningen ska använda i nästan alla fall
- Deklareras <u>utan</u> nyckelorder **static**

```
class Test1{
    int a = 0;

    void use(){
     System.out.println(a);
    }
}
```


Klassvariabler

- En klassvariabel är gemensam för alla instanser av en klass.
- Denna sorts variabler ska ni undvika
- Deklareras <u>med</u> nyckelorder **static**

```
class Tests2{
    static int a = 0;

    void use(){
     System.out.println(Test2.a);
    }
}
```

Funktioner då?

- Principen är densamma för funktioner
- Funktioner deklarerade utan **static** hör samman med (och opererar på) enskilda instanser, anropas med **objekt.funktionsNamn()**
- Funktioner deklarerade med **static** hör samman med (och opererar på) klassen, anropas med **KlassNamn.funktionsNamn()**
- Detta innebär i förlängningen att funktioner deklarerade med **static** inte kan komma åt variabler deklarerade utan **static** eftersom det inte är specificerat vilken instans instansvariabel som avses

Exempel?

- Kan du hitta på några egna exempel på klassmetoder och instansmetoder du använt?
- Klassmetoder **Math.max()**, **Math.round()**, alla era lincalcmetoder
- Instansmetoder **strängen.charAt()**, **objektet.equals()**, ...

Outline

Klasser och Objekt

Instansiering

Åtkomst

Calculator

Abstrakt datatyp

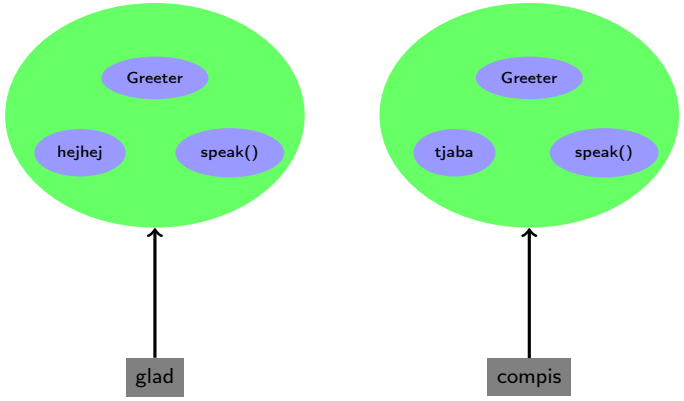
Att instansiera en klass

- 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

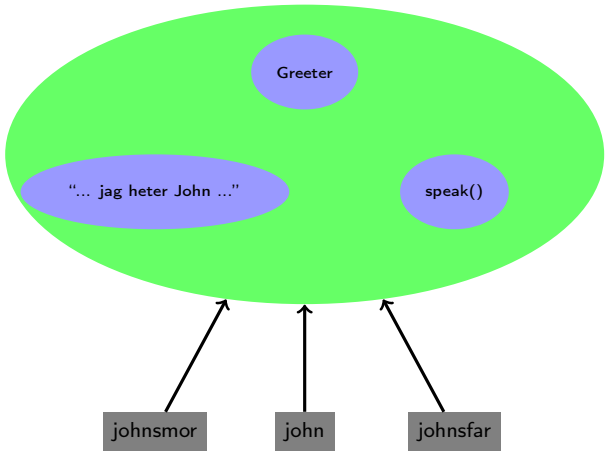
- En konstruktor saknar returtyp och heter samma som klassen
- Om ingen konstruktor skrivs så skapas automatiskt en så kallad standardkonstruktor som inte tar några argument
- Om man skapar en egen konstruktor så skapas inte standardkonstruktorn automatiskt
- Man kan ha godtyckligt många konstruktorer 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

Klasser och Objekt

Instansiering

Åtkomst

Calculator

Abstrakt datatyp

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

Klasser och Objekt

Instansiering

Åtkomst

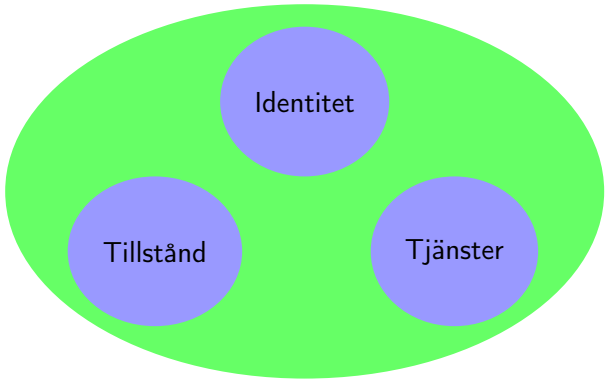
Calculator

Abstrakt datatyp

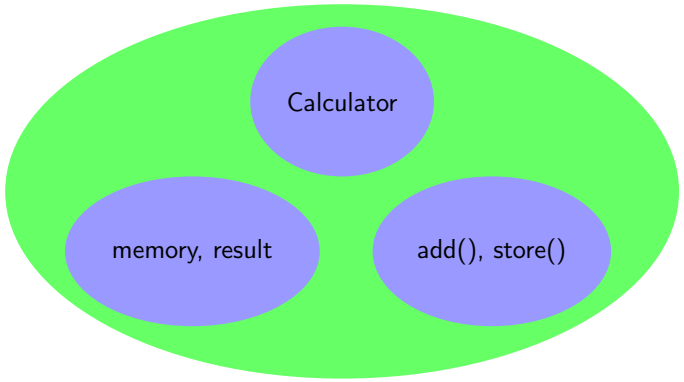
Calculator

```
class Calculator{
   private double memory, result;

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

   public String add(double c){
    return "hejhej";
   }

    public double getResult(){
     return result;
    }

    public void setResult(double _r){
     result = _r;
    }

   public void store(){
    memory = result;
   }
}
```

Ett objekt



Ett objekt



Använda kalkylatorn

```
Calculator calc = new Calculator();
calc.add(2.3, 5);
```

Flera konstruktörer till Calculator

```
public class Calculator {
    private double memory, result;
    public Calculator() {
     result = 0;
     memory = 0;
    }
    public Calculator(double loadedMem){
     result = 0;
     memory = loadedMem;
    }
 ...
```

Outline

Klasser och Objekt

Instansiering

Åtkomst

Calculator

Abstrakt datatyp

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

```java
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];
    }
}
```