



# Objects First With Java

## A Practical Introduction Using BlueJ

### Mer sofistikerat beteende

Att använda biblioteksklasser  
för att implementera  
en mer avancerad funktionalitet



# Vi skall gå igenom

- Hur man använder klassbiblioteket.
- Hur man läser dokumentation.
- Hur man skriver dokumentation.



# Javas klassbibliotek

- Tusentals klasser.
- Tiotusentals metoder.
- Många användbara klasser som underlättar programmeringen.
- En kompetent Java-programmerare måste känna till och kunna använda klassbiblioteket.



# Hur man arbetar med klassbiblioteket

Man skall:

- känna till vad de viktiga klasserna heter
- veta hur man hittar information om de andra klasserna

Poäng:

- Vi behöver bara känna till specifikationen, inte den faktiska implementeringen



# A Technical Support System

- Ett textdialogsystem
- Idén baseras på '*Eliza*' av Joseph Weizenbaum (MIT, 1960s)

låt oss undersöka det

# Huvudloopens struktur

```
boolean finished = false;
```

```
while(!finished) {
```

```
    gör något
```

```
    if(slutvillkoret) {
```

```
        finished = true;
```

```
    }
```

```
    else {
```

```
        gör något mer
```

```
    }
```

```
}
```

# Exempel

```
String input = reader.getInput();  
...  
String response = responder.generateResponse();  
System.out.println(response);
```



# Exempel

```
String input = reader.getInput();
```

```
if(input.startsWith("bye")) {  
    finished = true;  
}
```

- Var kommer 'startsWith' från?
- Finns inte i programmet.
- Vad är det för metod? Vad gör den?
- Hur kan vi ta reda på det?



# Hur man läser klassdokumentationen

- Dokumentation i HTML-format.
- Läses med en webbläsare.
- Java 2 Platform, Standard Edition, v 1.4.2 (1.5.0) API Specification
- *API: Application Programmers' Interface*
- Beskriver gränssnitten (interfaces) till alla Javas biblioteksklasser.



# Interface vs implementation

*Dokumentationen innefattar:*

- namnet på klassen;
- en övergripande beskrivning av klassen;
- information om klassens konstruktorer och metoder
- information om parametrar (och returvärden) för konstruktorer och metoder
- en beskrivning av avsikten med varje konstruktor och metod



**Klassens gränssnitt (interface)**



# Interface vs implementation

*Dokumentation innehåller **inte***

- fält som deklarerats *private* (de flesta fält är *private*)
- metoder som deklarerats *private* (hjälpmetoder)
- källkoden för metoderna



***implementationen av klassen***



# Användning av biblioteksklasser

- Biblioteksklasser måste importeras med importsatsen (utom klasserna i *java.lang*) till de klasser som använder dem.
- De kan sedan användas på samma sätt som de andra klasserna i det aktuella projektet.



# Import av paket

- Biblioteksklasser är organiserade i paket (*packages*).

- Enskilda klasser kan importeras:

```
import java.util.ArrayList;
```

- Alla klasser i ett paket kan importeras med:

```
import java.util.*;
```

# Sidospår: strängars likhet

```
if(input == "bye") {  
    ...  
}
```

*testar identitet*

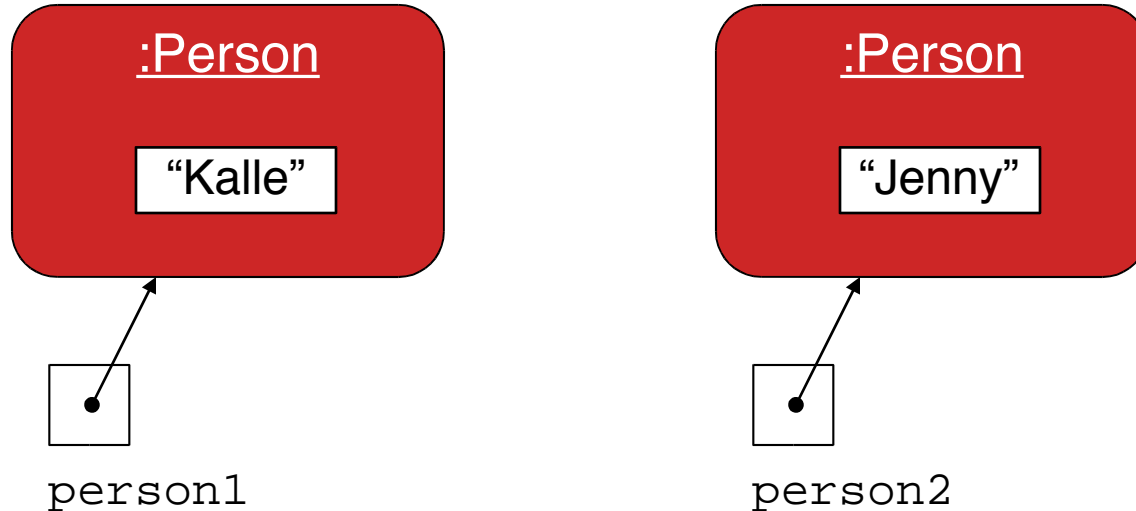
```
if(input.equals("bye")) {  
    ...  
}
```

*testar likhet*

- Strängar bör (nästan) alltid jämföras mha `.equals`

# Identitet vs likhet 1

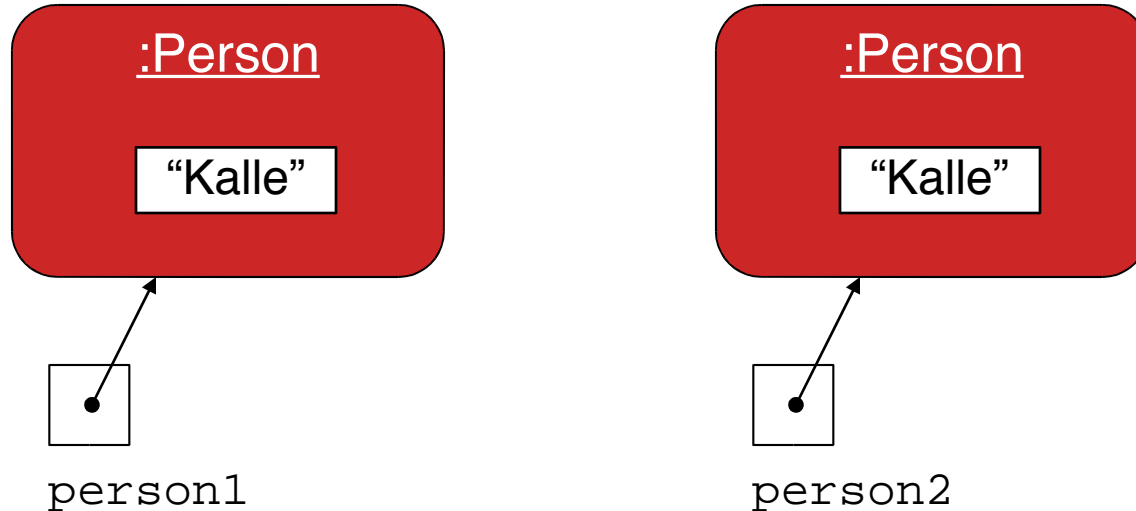
Övriga objekt (ej Strängar):



`person1 == person2 ?`

# Identitet vs likhet 2

Övriga objekt (ej Strängar):

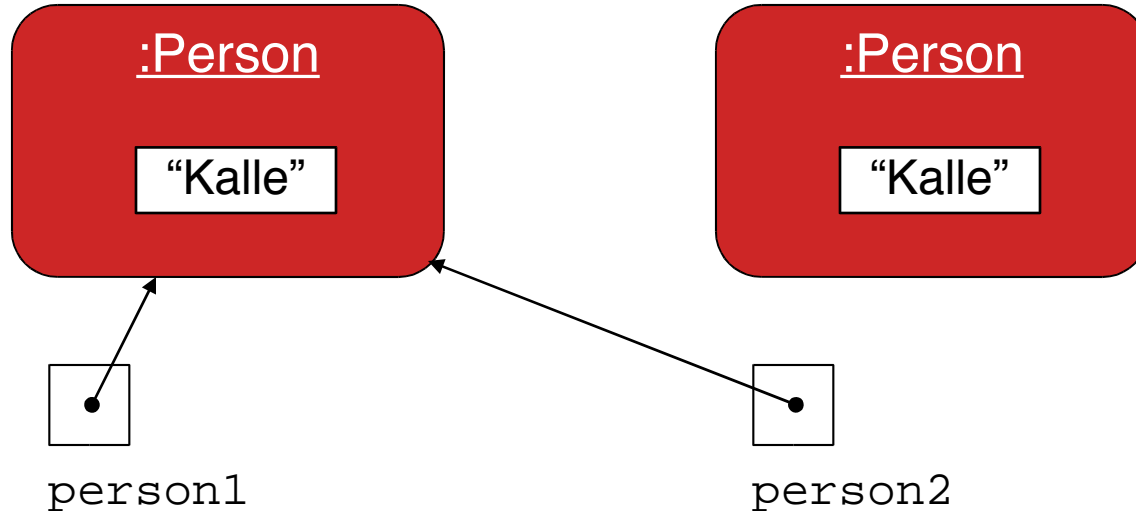


`person1 == person2 ?`



# Identitet vs likhet 3

Övriga objekt (ej Strängar):



`person1 == person2 ?`

# Identitet vs likhet (Strängar)

```
String input = reader.getInput();  
if(input == "bye") {  
    ...  
}
```

== testar identitet



Kan vara falsk!!!

# Identitet vs likhet (Strängar)

```
String input = reader.getInput();  
if(input.equals("bye")) {  
    ...  
}
```

**equals** testar  
likhet



# Random

- Biblioteksklassen `Random` används för att generera slumpetal

```
import java.util.Random;
...
Random randomGenerator = new Random();
...
int index1 = randomGenerator.nextInt();
int index2 = randomGenerator.nextInt(100);
```

# Generering av slumpmässiga svar

```
public Responder()  
{  
    randomGenerator = new Random();  
    responses = new ArrayList();  
    fillResponses();  
}  
  
public String generateResponse()  
{  
    int index = randomGenerator.nextInt(responses.size());  
    return (String) responses.get(index);  
}  
  
public void fillResponses()  
...
```

# Maps (sv. avbildningar)

- Maps är kollektioner som innehåller *par*.
- Ett par består av en nyckel (eng. *key*) och ett värde.
- Genom att man känner nyckeln så får man tillgång till värdet.
- Exempel: En telefonkatalog.



# En telefonkatalog

- Ett HashMap-objekt med nyckel och värde av typen String

:HashMap

"Karl Nilsson"	"(046) 939 458"
"Lisa Karlsson"	"(08) 536 464"
"Nils Svensson"	"(031) 488 123"

# HashMap

```
HashMap phoneBook = new HashMap();

phoneBook.put("Karl Nilsson", "(046) 939 458");
phoneBook.put("Lisa Karlsson", "(08) 536 464");
phoneBook.put("Nils Svensson", "(031) 488 123");

String number =
    (String)phoneBook.get("Lisa Karlsson");
System.out.println(number);
```



# Mängder (Sets)

```
import java.util.HashSet;  
import java.util.Iterator;  
...  
HashSet mySet = new HashSet();
```

```
mySet.add("one");  
mySet.add("two");  
mySet.add("three");
```

```
Iterator it = mySet.iterator();  
while(it.hasNext()) {  
    call it.next() to get the next object  
    do something with that object  
}
```

**Jämför med  
ArrayList!**

# Strängdelning

```
public HashSet getInput()  
{  
    System.out.print("> ");          // print prompt  
    String inputLine =  
        readInputLine().trim().toLowerCase();  
  
    String[] wordArray = inputLine.split(" ");  
  
    // add words from array into hashset  
    HashSet words = new HashSet();  
    for(int i = 0; i < wordArray.length; i++){  
        words.add(wordArray[i]);  
    }  
    return words;  
}
```

# Hur man skriver klassdokumentation

- Dina egna klasser bör dokumenteras lika väl som biblioteksklasser.
- Andra programmerare skall kunna använda dina klasser utan att behöva sätta sig in i implementationen.



# Dokumentationsregler

*Dokumentationen för en klass skall innehålla:*

- klassnamnet
- en kommentar som på ett övergripande sätt beskriver avsikten med klassen
- ett versionsnummer
- programmerarnas namn
- dokumentation för varje konstruktor och varje (public) metod



# Dokumentation av konstruktorer och metoder

*Dokumentationen för varje konstruktor och metod bör innehålla:*

- namnet på metoden
- returtypen
- parametrarnas namn och typ
- en beskrivning av avsikten med metoden och, om relevant, hur metoden fungerar
- en beskrivning av varje parameter
- en beskrivning av returvärdet (ev.)

# javadoc

## Klasskommentar:

```
/**
 * The Responder class represents a response
 * generator object. It is used to generate an
 * automatic response.
 *
 * @author      Michael Kölling and David J. Barnes
 * @version     1.0   (1.Feb.2002)
 */
```

# javadoc

## Metodkommentar:

```
/**
 * Read a line of text from standard input (the text
 * terminal), and return it as a set of words.
 *
 * @param  prompt  A prompt to print to screen.
 * @return  A set of Strings, where each String is
 *           one of the words typed by the user
 */
public HashSet getInput(String prompt)
{
    ...
}
```

# Public vs private

- Attribut som deklarerats *public* (fält, konstruktorer, metoder) är åtkomliga från andra klasser.
- Fält skall inte deklarerats *public*.
- Attribut som deklarerats *private* är inte åtkomliga från andra klasser utan bara från den egna klassen
- De metoder som är tänkta att användas av andra klasser skall deklarerats *public*. De andra skall vara *private*.



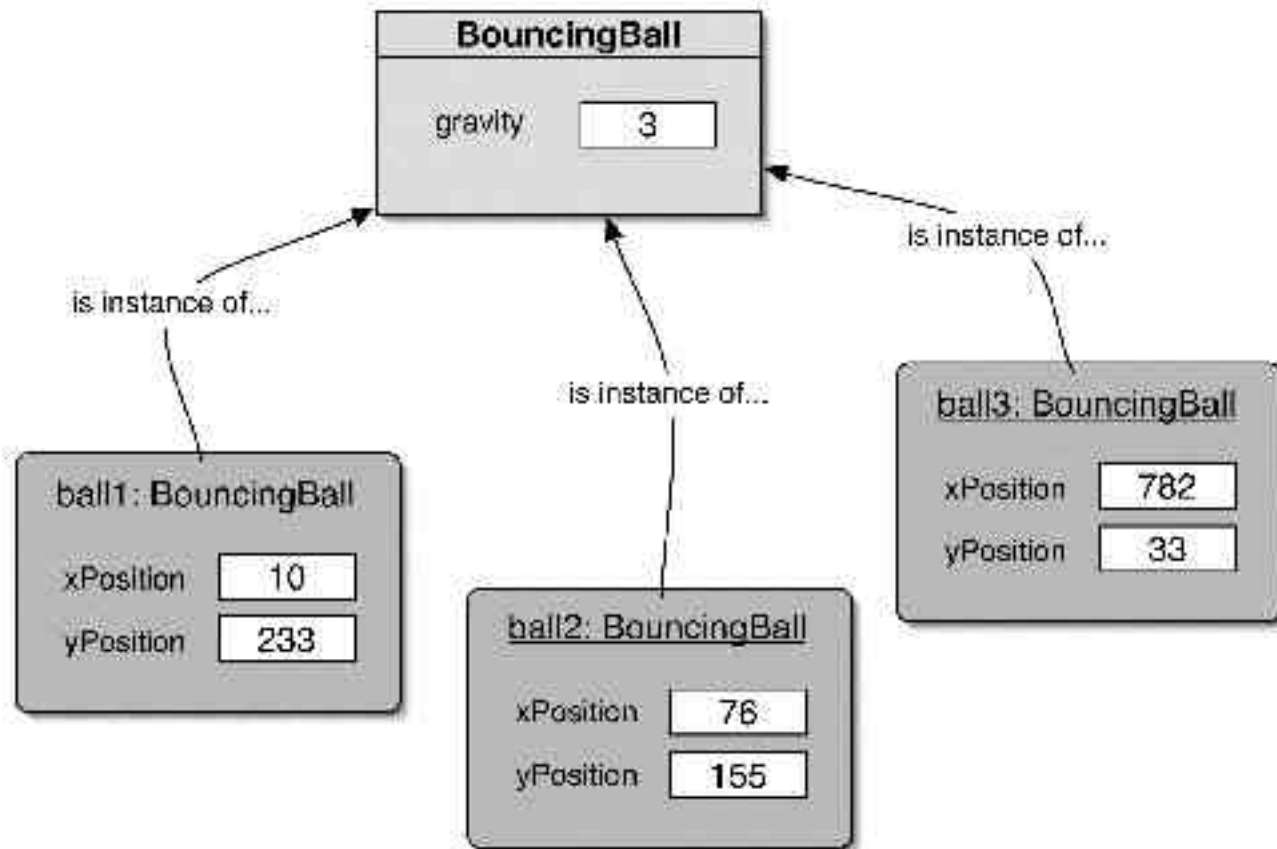


# Inkapsling

## *(information hiding)*

- Information som tillhör ett objekt skall döljas för andra objekt.
- Man skall veta vad ett objekt gör, inte hur det gör det.
- Inkapsling ökar graden av oberoende mellan delarna i ett program.
- Ett lågt beroende mellan delarna gör det lättare att underhålla och utveckla ett program.

# Klassvariabler



# Konstanter

```
private static final int gravity = 3;
```

- **private**: synlighetsmarkör (access modifier)
- **static**: klassvariabel (class variable)
- **final**: konstant



# Sammanfattning

- Java har ett omfattande klassbibliotek.
- Man måste känna till vad som finns i klassbiblioteket.
- Dokumentationen talar om det som vi behöver veta för att kunna använda en klass (interface).
- Implementationen visas inte (inkapsling).
- Vi dokumenterar egna klasser med hjälp av klasskommentarer och metodkommentarer så att javadoc kan producera en fullödlig dokumentation.

