Introduktion till OO

Andreina Francisco
(based on slides by Tobias Wrigstad)
What is object-oriented software

Objects are like people. They're living, breathing things that have knowledge inside them about how to do things and have memory inside them so they can remember things. And rather than interacting with them at a very low level, you interact with them at a very high level of abstraction [...]

Why Smalltalk?

I made up the term “object-oriented,” and I can tell you I did not have C++ in mind.

– Alan Kay

Alan Kay
Trivia: Smalltalk was the second "most loved programming language" in the Stack Overflow Developer Survey in 2017

```smalltalk
| i |
i := 5.
[i > 0] whileTrue:
    Transcript show: ((i*2) asString); cr.
    i := i - 1.
].

whileTrue: aBlock
    ^self value
    ifTrue: [
        aBlock value.
        [self value] whileTrue: [aBlock value].
    ].
```
Spädbarns sociala kompetens
Fler farmaceuter i vården
Innovationer inom life science
Professorn som skapar blixtar
Objekt

- Gäst A, Gäst B, Gäst C…
- Waiters …
- Kock A, …
- Tallrikar med mat
- Tables…
- Chairs…
- Tablecloths, glasses, silverware …
Objektorienteringens grundkoncept

Active and passive objects (Aktiva och passiva objekt)

Messaging (Meddelandesändning)

Aggregation (Aggregering)

Encapsulation (Inkapsling)

Inheritance (Arv)

Polymorphism (Polymorfism)
Objektorienteringens grundkoncept

Aktiva och passiva objekt

Aktiva: serveringspersonal, gäster, kockar
Passiva: maten, stolarna, borden, etc.

Meddelandesändning

Agregering

Inkapsling

Arv

Polymorfism
Objektorienteringens grundkoncept

Aktiva och passiva objekt

Meddelandesändning

Mellan aktiva objekt: beställa mat

Aktiva—passiva objekt: dra ut stol, äta, lyfta en gaffel

Aggregering

Inkapsling

Arv

Polymorfism
Objektorienteringens grundkoncept

Aktiva och passiva objekt

Meddelandesändning

Aggregering

   Ett bord består av en bordsskiva och fyra ben

   Ett middagssällskap består av flera gäster

Inkapsling

Arv

Polymorfism
Objektorienteringens grundkoncept

Aktiva och passiva objekt

Meddelandesändning

Aggregering

Inkapsling

A waiter can serve the customers but can’t set their mood to “happy”

Hur maten lagas (Det är inte uppenbart att det är hästkött i lasagnen, jmf. abstraktion)

Arv

Polymorfism
Objektorienteringens grundkoncept

Aktiva och passiva objekt

Meddelandesändning

Aggregering

Inkapsling

Arv

En Gäst är en Person, en Kypare är en Person, en Kock är en Person

Om $\mathcal{P}(\text{Person}) \implies \mathcal{P}(\text{Gäst}) \land \mathcal{P}(\text{Kypare}) \land \mathcal{P}(\text{Kock})$

Polymorfism
Objektorienteringens grundkoncept

Aktiva och passiva objekt
Meddelandesändning
Aggregering
Inkapsling
Arv
Polymorfism

Different objects can have the same interface

Different dishes taste different, people act different, etc.

The chefs also go to a restaurant as guests, you can drink wine from beer glasses
Playing The Sims
Java Bootstrap

Two labs

Dice

Cash register simulator  (note that you get to report goals on this task!)

Deadline for both labs: 25/10

Continue with assignments 3 and 4

Already published

Warning! Assignment 4 has been pushed out quickly! Please, make suggestions for improvements!
Concepts

Object

Class – descriptions of objects
Concept

Object

The world consists of objects (which consist of objects ...) that send messages to each other.

Objects "of the same kind" are grouped into classes that describe how the objects work.

Object relationships: aggregation (objects have references to other objects).

An object's "building blocks" are not directly accessible (encapsulation).

Class - descriptions of objects
**Konzept**

**Object**

The world consists of objects (which consist of objects ...) that send **messages** to each other

Objects "of the same kind" are grouped into **classes** that describe how the objects work

Object relationships: **aggregation** (objects have references to other objects)

An object's "building blocks" are not directly accessible (**encapsulation**)

**Class - descriptions of objects**

Describes not only what an object contains (**state**) but also its **behaviour**

Relationships between classes: **inheritance** (A poodle is a dog, which is an animal, which is a...)

A class’ internal structure is not visible from the outside (**encapsulation**)
Procedurell programmering

\[ f(x) \] — du bestämmer ”nu skall jag göra f på datat \( x \)”

Objektorienterad programmering

\[ x.f() \] — du ber ”snälla \( \text{objekt} \) \( x \), utför f”
Statisk bindning i C

\[ f(x) \] — *gcc* selects \( f \) depending on the type of \( x \) when *compiling*.

Dynamisk bindning i Java

\[ x.f() \] — *The VM* selects \( f \) depending on \( x \) at *runtime*! 
Introduktion till OOP med Java
En liten Java-parlör (stämmer för de flesta OOPLs)

<table>
<thead>
<tr>
<th>Svenska</th>
<th>Engelska</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objekt</td>
<td>Object</td>
</tr>
<tr>
<td>Klass</td>
<td>Class</td>
</tr>
<tr>
<td>Arv</td>
<td>Inheritance</td>
</tr>
<tr>
<td>Instansvariabel / fält</td>
<td>Instance variable / field</td>
</tr>
<tr>
<td>Metod</td>
<td>Method</td>
</tr>
<tr>
<td>Superklass / basklass</td>
<td>Super class / base class</td>
</tr>
<tr>
<td>Subklass / härledd klass</td>
<td>Sub class / derived class</td>
</tr>
<tr>
<td>Abstrakt klass</td>
<td>Abstract class</td>
</tr>
<tr>
<td>Superanrop</td>
<td>Super call</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Svenska</th>
<th>Engelska</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metodspecialisering</td>
<td>Overriding</td>
</tr>
<tr>
<td>Överlagring</td>
<td>Overloading</td>
</tr>
<tr>
<td>Överskuggning</td>
<td>Shadowing</td>
</tr>
<tr>
<td>Klasshierarki</td>
<td>Class hierarchy</td>
</tr>
<tr>
<td>Aggregering</td>
<td>Aggregation</td>
</tr>
<tr>
<td>Typomvandling</td>
<td>Type cast</td>
</tr>
<tr>
<td>Polymorfism</td>
<td>Polymorphism</td>
</tr>
<tr>
<td>Barnklass</td>
<td>Child / sub / derived class</td>
</tr>
<tr>
<td>Instantieras</td>
<td>Instantiate</td>
</tr>
</tbody>
</table>
Interpreting the Java Compiler Error Messages!

Template

Example

CommonCompilerErrors.java:66: error: cannot find symbol
  LinkedList myList;
  ^
  symbol:   class LinkedList
  location: class ErrorThree
Förstå kompilatorns språk

<table>
<thead>
<tr>
<th>Kompilatorn säger</th>
<th>Betyder i regel</th>
</tr>
</thead>
<tbody>
<tr>
<td>cannot find symbol</td>
<td>Felstavat namn, eller namnet är inte synligt ännu, t.ex. inte importerat in, alt finns inte</td>
</tr>
<tr>
<td>method X cannot be applied</td>
<td>Argumentlistans typer fel (för få argument, fel ordning, fel argument?)</td>
</tr>
<tr>
<td>incompatible types</td>
<td>Typen på högersidan är inte kompatibel med den till vänster. Är de subtyper?</td>
</tr>
<tr>
<td>X cannot be converted to String</td>
<td>Glömt att anropa toString()?</td>
</tr>
</tbody>
</table>
Object

- A collection of data and operations that manipulate that data
- You can send messages to an object
  The object selects what to do as a response
- Object-oriented design is data-driven design
  Which actors are there?
  How are they related to inheritance, aggregation, usage, etc. (more later)
Class (finns i nästan alla OO-språk)

- A class is a "model" from which you can build infinitely many objects

- Membership
  - Instance variables (also fields)
  - Methods

- A class is like a structure + all functions that operate on the structure

- Things we'll talk about later
  - Relationships between classes
    - Access modifiers
    - Inheritance

- **Instanciation**: to create an object from a class
Java

- Developed by Sun (James Gosling) in the early 90's, released in 1995
- Some design principles for Java
  - Simple, object oriented and friendly
  - Robust and safe
  - Architecture independent and portable
  - Quick
  - Clear
    - Class Foo must be in Foo.java
Ett första Java-program

```java
/**
 * @author Tobias Wrigstad (tobias.wrigstad@it.uu.se)
 * @date 2013-10-01
 */
public class Hello {
    String who = null;
    public Hello (String who) {
        this.who = who;
    }
    public void greet() {
        System.out.println("Hello " + who);
    }
    public static void main(String args[]) {
        if (args.length > 0) {
            Hello hello = new Hello(args[0]);
            hello.greet();
        } else {
            System.out.println("Usage: java Hello <who>");
        }
    }
}
```
public class Hello {
    String who = null;
    public Hello (String who) {
        this.who = who;
    }
    public void greet() {
        System.out.println("Hello " + who);
    }
    public static void main(String args[]) {
        if (args.length > 0) {
            Hello hello = new Hello(args[0]);
            hello.greet();
        } else {
            System.out.println("Usage: java Hello <who>");
        }
    }
}
Compile and run!

- Compiler “javac”
  - Understands dependencies
  - Compile to “Java-bytecode”
- The program must run in the virtual machine
  - The program “java”
  - Takes as argument the name of a class with a main method

$ javac Hello.java
$ ls
Hello.java
Hello.class
$ java Hello
Usage: java Hello <name>
$ java Hello "Tobias"
Hello Tobias!
$
Länkad lista i C

Each link is a “mallocated” structure

A link points to its next link

A set of functions that operate on all parts of the list

value 4
next

value 5
next

value 9
next NULL

list

first
last

Länkad lista i Java

Each link is an Object

Each link refers to its next link

The List and the Link Objects have separate permissions and define their own behaviours

first

next

next

next

first

next

next

next

last

next

next

next

List

Link

Link

Link
Observation!

- Two classes: List and Link

  List-object aggregates Link-objects

  Recursion and iteration like in C (Note that the recursive call switching receivers!)

- Private and Public access

  Requires set and get methods in Link for private member variables
References vs Pointers

- A reference is a handle to an object - it is not an address to a place in memory
  - All valid pointers in C do not point to what they should (or anything at all)
  - All references in Java always point to what they should and to something!
- The reference null is not the address 0
  - It is also not a Boolean value!!
- References enable automatic memory management (GC)
Automatic Memory Management

- Java handles memory automatically
  
  `new ClassName(...)` automatically allocates enough memory to the heap
  
  When the last reference to an object is removed, the object is junk
  
  When the memory becomes full, junk is automatically cleared to leave space for new items
  
- Also:

  No `malloc` (new always allocate on the heap, at least what you know!)
  
  No `free`
Treasonably similar to C!

- The syntax is chosen to make it easy for C and C++ programmers to program Java
- Many conceptual differences (Java is more like Smalltalk than C++)

**BUT:** we can take with us a lot from C!

While, for, if, switch, variable declarations, function syntax, primitive types, etc.….  

By and large, you already know how to program Java, just *not how to program object-oriented in Java*!

- Be careful not to program C in Java!