

DE4 - Object Oriented Programming and Architecture  
Spring 2008

Daniel Ortiz-Arroyo  
Computer Science Department  
Aalborg University Esbjerg

3.0

---

---

---

---

---

---

---

---

About myself

- Associate Professor at AAUE's Computer Science Department
- Research on
  - Computational Intelligence
  - High Performance and Distributed Computing
- Usually teach in both DAT/FxS and DE educations
- Better to contact me by email ([do@cs.aaue.dk](mailto:do@cs.aaue.dk))
- Web page: [www.cs.aaue.dk/~do](http://www.cs.aaue.dk/~do) with links to this course's web page in the "Teaching" section

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo

2

---

---

---

---

---

---

---

---

About the course

- This is an *introductory* course on object-oriented (OO) programming and architecture (more programming than architecture)
- 10 Lectures (+ 5 of self study)
- We will use Java as an example of OO language
  - We won't cover *all details* of Java and its libraries but only an important subset
- I assume you know C language so I'll go very quickly on the issues that are similar to Java
- We will discuss some theory on OO systems but focus more on the design and coding stages of software development

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo

3

---

---

---

---

---

---

---

---

## About the course

- Course will be interactive:
  - lectures + exercises combined at lecture room
- Bring your laptop next class
  - Please coordinate yourselves. You can work in groups of 2 so not all of you need to bring your laptops
  - We are likely to do programming exercises every lecture
- Bring your text book all times too, because the exercises will be mostly taken from text book

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo

4

---

---

---

---

---

---

---

---

## Course Schedule

Lecture:	Date:	Topic:	Literature:
1	4/3	Introduction: Procedural vs OOP: Objects, Classes, Methods, Attributes. Lays and BlueJ. Understanding class definitions, accessors and mutators.	Chapters 1,2
2	5/3	Object interaction: abstraction and modularization. Class and object diagrams. Multiple constructors. Method calls. Debugger. Examples.	Chapter 3
3	11/2	Collections, generic classes, loops, iterators and arrays.	Chapter 4
4	12/2	Library classes: Random numbers, Packages, Maps and sets. Information hiding.	Chapter 5
5	17/3	Object Oriented Design: Designing classes and applications.	Chapters 7, 13
6	25/3	Inheritance and polymorphism.	Chapters 8,9
7	26/3	Abstract classes, Multiple inheritance, Interfaces.	Chapter 10
8	1/3	Building GUIs: Examples.	Chapter 11
9	2/4	Basic testing and debugging: techniques. Handling errors.	Chapters 6,12
10	9/4	Design patterns. Case study.	Chapters 13,14

This is an ambitious schedule!! We'll cover most of the text book

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo

5

---

---

---

---


---

---

---

---

## Books

Textbook: *Objects First with Java: A Practical Introduction using BlueJ*. David J. Barnes & Michael Kölling, Pearson Education, 2006 ISBN 0-13-197629-X. (Available at AAUE Bookstore) 

References:

1. *Java How to Program* by H.M Deitel and P.J. Deitel Pearson Education, 2006 ISBN 0-13-129014-2
2. *Thinking in Java* by Bruce Eckel available electronically at: <http://www.ibiblio.org/pub/docs/books/eckel/>
3. *Head First Design Patterns* by Elisabeth Freeman, Eric Freeman, Bert Bates, Kathy Sierra, O'Reilly Media

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo

6

---

---

---

---

---

---

---

---

## Goals of the course

- Getting sound knowledge of
  - programming principles
  - object-orientation
- By the end of the course you must be able to
  - implement a small software system in Java
  - understand and use some of Java's libraries
  - understand basic UML diagrams
  - understand basic OO design techniques and a few design patterns

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo

7

---

---

---

---

---

---

---

---

## Course overview (1)

- Objects and classes
- Understanding class definitions
- Object interaction
- Grouping objects
- More sophisticated behavior - libraries
- Well-behaved objects - testing, maintaining, debugging
- Designing classes

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo

8

---

---

---

---

---

---

---

---

## Course overview (2)

- Inheritance
- Polymorphism
- Extendable, flexible class structures
- Designing applications
- Basics of Design Patterns
- Building graphical user interfaces
- Handling of errors

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo

9

---

---

---

---

---

---

---

---

## Software Design

The diagram shows a 'Real World Problem' in a thought bubble. An arrow labeled 'Abstraction' points from the problem to a box containing 'Model' and 'Algorithm', which are connected by a double-headed arrow. This box is labeled 'Software System'. A return arrow labeled 'Interpretation' points from the 'Software System' back to the 'Real World Problem'. Below the diagram, a smiley face icon is followed by the text: '•Real world objects can be modeled using programming abstractions called *objects*'.

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo 10

---

---

---

---

---

---

---

---

## Procedural vs OO thinking

The diagram compares two ways of solving a 'Real World Problem'. On the left, a smiley face icon is connected to a code block: 

```
int x=10,y=100;
struct{...}
void main(){
  proc1();
  proc2();
}
```

 On the right, another smiley face icon is connected to a diagram of three objects: 'Object A', 'Object B', and 'Object C'. 'Object A' and 'Object B' are connected by a double-headed arrow, and 'Object A' and 'Object C' are also connected by a double-headed arrow. Below the diagram, the text reads: 'OO provides a higher level of abstraction to a problem in terms of objects'.

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo 11

---

---

---

---

---

---

---

---

## Procedural vs OO thinking

The diagram compares two hierarchical structures. On the left, a tree diagram shows 'main' at the top, branching into 'procedure1' and 'procedure2'. 'procedure1' branches into 'procedure5' and 'procedure3'. 'procedure2' branches into 'procedure3' and 'procedure4'. Below this is the text: 'A hierarchy of procedure calls'. On the right, a tree diagram shows 'Class A' at the top, branching into 'Class B' and 'Class C'. Below this is the text: 'OO a hierarchy of classes'.

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo 12

---

---

---

---

---

---

---

---

## Buzzwords in OO systems

responsibility-driven design

inheritance                      encapsulation

iterators                      overriding                      coupling

cohesion                      interface

collection classes                      mutator methods

polymorphic method calls

design patterns

*Goal by end of course is that you understand what all these words mean*

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo      13

---

---

---

---

---

---

---

---

## Fundamental concepts

- class
- object
- method
- parameter
- data type

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo      14

---

---

---

---

---

---

---

---

## Objects and classes

- objects
  - represent 'things' from the real world, or from some problem domain (example: "the red car down there in the car park")
    - objects are dynamic
- classes
  - represent all objects of a kind (example: "car" is a general concept)
    - classes are static

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo      15

---

---

---

---

---

---

---

---

## Methods and parameters

- Objects have operations which can be invoked (Java calls them *methods*).
- Methods may have parameters to pass additional information needed to execute them.

Java's methods, are similar to procedures and functions in C language

```
object  
method1(param,param);  
method2(param);  
method3();
```

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo

16

---

---

---

---

---

---

---

---

## Other observations

- Many *instances* can be created from a single class. These instances are called *objects*.

Program

```
class X{  
...  
}
```

class is a static declaration in the source code



Objects of class X



objects are dynamic, they are created at run time

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo

17

---

---

---

---

---

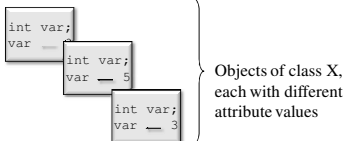
---

---

---

## Other observations

- An object has *attributes*: values stored in *fields*.
- The class defines what fields an object has, but each object stores its own set of values (i.e. the *state* of the object).



Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo

18

---

---

---

---

---

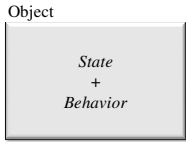
---

---

---

## State and behavior

- In summary an object has
  - *State* (value of attributes) and
  - *Behavior* (methods used to process and interact with the object's attributes)



The diagram shows a rectangular box labeled "Object" at the top. Inside the box, the word "State" is written above a plus sign "+", which is above the word "Behavior".

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo 19

---

---

---

---

---

---

---

---

## Object Oriented Languages

- Programming models that include objects and classes are built in many languages today
  - Java
  - C++
  - C# (similar to Java for MS .NET)
  - J# (similar to Java for MS .NET)
  - Python
  - Delphi
  - Small Talk
- OOP is the dominant paradigm today to build complex software systems
- Simula (Dept. Informatics, Univ. Oslo, Norway) was the first OO language in 1960s

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo 20

---

---

---

---

---

---

---

---

## Java and OOP

- A popular OO language created by James Gosling in Sun Microsystems
- We'll discuss OO features of Java. The *concepts* you will learn can be applied to other OO languages.
- Java has grown from an early implementation for embedded systems to include libraries for distributed systems, GUIs, cell phones, and many other applications.

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo 21

---

---

---

---

---

---

---

---

## Definition of a Class in Java

```
public class Circle
{
    private int diameter;
    private int xPosition;
    private int yPosition;
    private String color;
    private isVisible;

    public void method1() {...}
    public void method2() {...}
}
```

*attributes/state*

*methods/behavior*

Each class has source code associated with it that defines its details (fields/attributes and methods).

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo 22

---

---

---

---

---

---

---

---

## Return values

- Methods may return a result via a return value.
- In general values returned can be primitive types (e.g. int, float) or objects
  - As we mentioned, methods are similar to procedures in C except that they can return also *objects*

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo 23

---

---

---

---

---

---

---

---

## Break

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo 24

---

---

---

---

---

---

---

---

## Bluej

- Bluej is the software tool that we will use in the class, it's included in the CD's textbook
  - Can be downloaded at <http://www.bluej.org/>
  - It's a *basic* Integrated Development Environment (IDE) *only used for teaching*
  - For *real development* we should use other more complete/sophisticated IDE's (most are free)
    - Netbeans
    - JCreator
    - Borland JBuilder
    - Eclipse

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo 25

---

---

---

---

---

---

---

---

## Bluej

- Demo of Bluej

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo 26

---

---

---

---

---

---

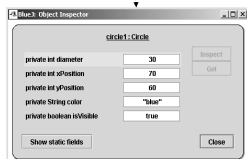
---

---

## Objects/State

Class Circle

```
//Create an instance of class circle (an object)
//in the program
Circle circle1= new Circle();//details later
```



Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo 27

---

---

---

---

---

---

---

---

## Two circle objects

```
//Create two instances of class circle
Circle circle1= new Circle();
Circle circle2= new Circle();
```

Circle

- int diameter
- int xPositon
- int yPosition
- String color
- boolean isVisible

circle\_1: Circle

- diameter 50
- xPositon 80
- yPosition 30
- color "blue"
- isVisible true

circle\_2: Circle

- diameter 30
- xPositon 230
- yPosition 75
- color "red"
- isVisible true

Attribute values change

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo 28

---

---

---

---

---

---

---

---

## Problem example: Ticket machines - an external view

- Exploring the behavior of a typical ticket machine.
  - *Naive-ticket-machine* project is included in the textbook's CD.
  - Machines supply tickets of a fixed price.
    - How is that price determined?
  - How is 'money' entered into a machine?
  - How does a machine keep track of the money that is entered?

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo 29

---

---

---

---

---

---

---

---

## Ticket machines - an internal view

- Interacting with an object gives us clues about its behavior.
- Looking inside allows us to determine how that behavior is provided or implemented.
- All Java classes have a similar-looking internal view.

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo 30

---

---

---

---

---

---

---

---

### Basic class structure

```
public class TicketMachine  
{  
    Inner part of the class omitted.  
}
```

The outer wrapper of TicketMachine

```
public class ClassName  
{  
    Fields  
    Constructors  
    Methods  
}
```

The contents of a class

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo 31

---

---

---

---

---

---

---

---

### Fields

- Fields store values for an object.
- They are also known as instance variables.
- In Bluej Use the *Inspect* option to view an object's fields.
- Fields define the state of an object.

```
public class TicketMachine  
{  
    private int price;  
    private int balance;  
    private int total;  
    Further details omitted.  
}
```

visibility modifier    type    variable name

private int price;

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo 32

---

---

---

---

---

---

---

---

### Constructors

- Constructors initialize an object.
- They have the same name as their class.
- They store initial values into the fields.
- They often receive external parameter values for this.

```
public class TicketMachine{  
    ...  
    public TicketMachine(int ticketCost)  
    {  
        price = ticketCost;  
        balance = 0;  
        total = 0;  
    }  
    ...  
}
```

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo 33

---

---

---

---

---

---

---

---

## Passing data via parameters

calls the constructor

34

---

---

---

---

---

---

---

---

## Assignment

- Values are stored into fields (and other variables) via assignment statements:
  - `variable = expression;`
  - `price = ticketCost;`
- A variable stores a single value, so any previous value is lost (same as in C)

35

---

---

---

---

---

---

---

---

## Accessor methods

- Methods implement the *behavior* of objects.
- Accessors are methods that provide information about an object.
- Methods have a structure consisting of a header and a body (same as in C)
- The header defines the method's *signature* (i.e. how it should be called)  
`public int getPrice()`
- The body encloses the method's statements.

36

---

---

---

---

---

---

---

---

## Accessor methods

```
public int getPrice()  
{  
    return price;  
}
```

Labels: visibility modifier, return type, method name, parameter list (empty), return statement, start and end of method body (block).

Java methods look similar to procedures in C except that they include a visibility modifier declaration

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo 37

---

---

---

---

---

---

---

---

## Mutator methods

- Have a similar method structure: header and body.
- Used to *mutate* (i.e., change) an object's state.
- Achieved through changing the value of one or more fields.
  - Typically contain assignment statements.
  - Typically receive parameters.

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo 38

---

---

---

---

---

---

---

---

## Mutator methods

```
public void insertMoney(int amount)  
{  
    balance = balance + amount;  
}
```

Labels: visibility modifier, return type, method name, parameter, field being mutated, assignment statement.

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo 39

---

---

---

---

---

---

---

---

## Get/Set Methods

- Accessor and mutator methods are also usually named get/set methods

```
private int value; //visible only inside
public void setValue(int new_value)
{
    value = new_value; //set (mutator)
}
public int getValue()
{
    return value; //get (accessor)
}
```

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo

40

---

---

---

---

---

---

---

---

## Printing from methods

```
public void printTicket ()
{
    // Simulate the printing of a ticket.
    System.out.println("#####");
    System.out.println("# The BlueJ Line");
    System.out.println("# Ticket");
    System.out.println("# " + price + " cents.");
    System.out.println("#####");
    System.out.println();

    // Update the total collected with the balance.
    total = total + balance;
    // Clear the balance.
    balance = 0;
}
```

Comments in Java

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo

41

---

---

---

---

---

---

---

---

## C and Java methods

- In summary methods in Java differ from procedures and functions in C in the following:
  - Java methods may return objects
  - Java methods include a visibility modifier

Slides by David J. Barnes, Michael Kölling modified by Daniel Ortiz-Arroyo

42

---

---

---

---

---

---

---

---