Object-Oriented Programming

General information
Topic 1: Introduction and the concept of objects

Anders Jonsson & Federico Heras
2023-24

Theory session 1

General information

Topic 1: Introduction and the concept of objects

Object-oriented programming

General information

Responsible teachers (coordinators)

- Anders Jonsson (group 1)
  - Email: anders.jonsson@upf.edu
  - Tutoring: by appointment
- Federico Heras (group 2)
  - Email: federico.heras@upf.edu
  - Tutoring: by appointment

General information

Organization:

- 12 theory sessions
- 6 lab sessions
- 5 seminar sessions
- Individual programming exercises

Communication:

- Through Aula Global
  - Slides from theory sessions
  - Handouts with lab and seminar exercises
  - Message board
  - etc.

Evaluation

Theory:

- A final exam at the end of the trimester
- A makeup exam in July (only for those who do not pass)

Labs:

- Five lab solution deliveries (1-2 weeks after class)

Seminars:

- Five seminar solution deliveries (at the end of each class)

Evaluation

The final grade is calculated as follows:

- Final exam: 35%
- Five labs: $5 \times 8\% = 40\%$
- Five seminars: $5 \times 4\% = 20\%$
- Programming exercises: 5%

To pass the course it is necessary to

- pass the exam with a grade ≥ 5
- pass the labs with an average grade ≥ 5
- pass the seminars with an average grade ≥ 5

Seminars

- Objective: solve exercises related to design and modelling
- Prior preparation (revise theory concepts, read handout)
- Solve one or several concrete problems
- In classroom, conceptual work without writing code
- The solution is developed in class
- Solution delivery at the end of class
- Work in pairs

Labs

- Objective: implement (in Java) the solutions to seminars
- Solution to previous seminar handed back at beginning of class
- Necessary techniques are explained
- On computer, solution implemented during class and at home
- Delivery at a later date, before next lab session
- Delivery: Java code + report
- Work in pairs

Programming exercises

- Objective: strengthen programming ability
- Solve a series of programming exercises
- https://www.hackerrank.com/
- Solution implemented at home
- Individual work

Teachers

Group	X=1	X=2
TX	Anders Jonsson	Federico Heras
PX01	Ahana Deb	Patricia Carbajo
PX02	Pritam Mishra	Anders Jonsson
PX03	–	Patricia Carbajo
SX01	Ahana Deb	Patricia Carbajo
SX02	Carlos Giraldos	Ramón González Castillo
SX03	Pritam Mishra	Patricia Carbajo
SX04	Anders Jonsson	Ramón González Castillo

Course topics

Topic 1 Introduction and the concept of objects

Topic 2 The object-oriented programming paradigm

Topic 3 Object modelling and relations between objects

Topic 4 Inheritance and polymorphism

Topic 5 Abstract classes and interfaces

Topic 6 Reuse and study of problems solved using objects

Bibliography

- Bertrand Meyer: *Object Oriented Software Construction*
- Cay Horstmann: *Object Oriented Design and Patterns*
- Bruce Eckel: *Thinking in Java*
- Ken Arnold, James Gosling, David Holmes: *Java Programming Language*
- Slides from theory sessions

Theory session 1

General information

Topic 1: Introduction and the concept of objects

Object-oriented programming

What is “programming”?

What is "programming"?

Wikipedia:

- "Programming involves tasks such as analysis, generating algorithms, profiling algorithms' accuracy and resource consumption, and the implementation of algorithms [...] in a chosen programming language"
- "The purpose of programming is to find a sequence of instructions that will automate the performance of a task"

Language incompatibility

Evolution of programming languages

1940 Program: sequence of 0's and 1's

1951 Assembly language: short words

1955 FORTRAN: first imperative language

1958 LISP: first functional language

1967 Simula: first object-oriented language

1972 Prolog: first logical language

1973 C: efficient translation to machine code

1980 C++: imperative, object-oriented language

1991 Python: object-oriented language that emphasizes readability

1995 Java: object-oriented, multi-platform language

Code examples

- Modern language:

```
x = 23 + 42  (Python)
x = 23 + 42; (C, C++, Java)
```

- Assembly:

```
LDA #23
ADD #42
STO 34
```

- Machine code:

```
0001 00010111
0100 00101010
0000 00100010
```

Motivation for progress

- Fundamentally, facilitate the task of writing programs
- Bring the programming language closer to natural language
- Promote abstraction
- Promote reuse

Abstraction

- "The process of removing physical, spatial, or temporal details or attributes in the study of objects or systems to focus attention on details of greater importance"
- Focus on the essential
- Hide what is irrelevant
- Important tool for reducing the complexity of a problem

Reuse

- "Action or practice of using an item [...] to fulfil a different function"
- The ability to reuse relies [...] on the ability to build larger things from smaller parts
- Take advantage of existing elements
- Avoid duplicating the effort needed to create a new element
- Important tool for reducing the work effort

Evolution towards objects

1. Procedures and functions
2. Modules (libraries, packages)
3. Abstract data types
4. Objects

Procedures and functions

- Group sequences of instructions into individual actions
- Implement concrete tasks
- Example (Python):

```
def sum( a, b ):
    s = a + b
    return s
```

- Example (C):

```
int sum( int a, int b ) {
    int s = a + b;
    return s;
}
```

Modules (libraries, packages, etc.)

- Group procedures and functions
- Implement sets of related tasks
- Publish the definition of procedures and functions, but hide the implementation
- Example:

```
import math

    math.sqrt(x)
        Return the square root of x.

    math.cos(x)
        Return the cosine of x radians.
```

Abstract data types

- Associate data types with operations
- Abstract: conceptual description instead of concrete implementation
- Example: Stack
  - `push(S,e)`: add an element `e` to the stack `S`
  - `pop(S)`: remove the top element from the stack `S`
  - `peek(S)`: access the top element in the stack `S`

Objects

- Also associate entities with characteristics and behavior
- In addition, incorporate novel concepts
- Objective: increase the level of abstraction and reuse

Object-oriented programming languages

- Java, C++, Python, Ruby, C#, R, PHP, Visual Basic.NET, JavaScript, Perl, SIMSCRIPT, Object Pascal, Objective-C, Dart, Swift, Scala, Kotlin, Common Lisp, MATLAB, Smalltalk

Applications of OOP

mozilla Firefox

C++

Python

Java

Ruby

Why program with objects?

Why program with objects?

Why program with objects?

Why program with objects?

Look at the C program "football.c"

Think about how to change the program in the following ways:

- Add player positions (e.g. right defender etc.)
- Add several league and cup competitions
- Adapt different league rules from different countries
- Add the possibility to simulate matches
- Add female players and female leagues
- Add national teams

Problems with programs that are not object-oriented

- Harder to make programs modular (made of small pieces)
- Related information is not stored together
  (e.g. the number of goals of a player is stored in a different data structure than the name)
- No direct references between different concepts
  (e.g. to list the players of a team, we have to go through a list of indices and look up the players in a different data structure)
- More difficult to make changes and add novel features
- Many unrelated functions appear together
- Program files are large
- Use of global variables

What do we mean by "object"?

Utility of objects

- Objects help us interact with our surroundings
- Important tool for abstraction
- Group objects by *type*



What information do objects convey?

What information do objects convey?

- Two fundamental categories of information:
  - **Descriptive**: rectangular, light, black, etc.
  - **Functional**: call, listen, play, watch, etc.

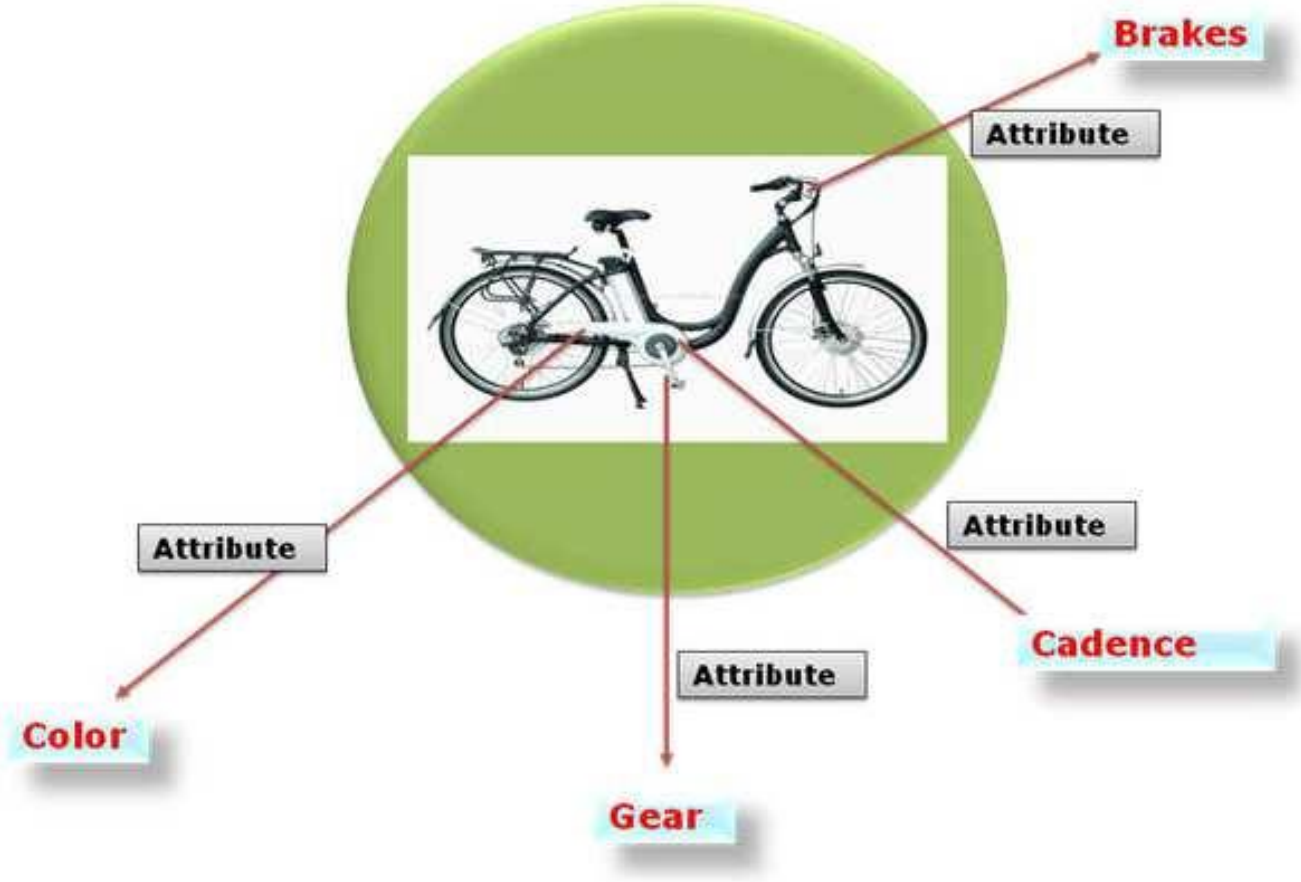
How can we represent objects in a computer?

- A computer limits our capacity to represent physical objects
- Idea: represent essence of an object (information it conveys)
- Object represented by the two categories of information that describe it:
  - Attributes (or variables) that describe characteristics
  - Methods (or functions) that implement behavior

Example

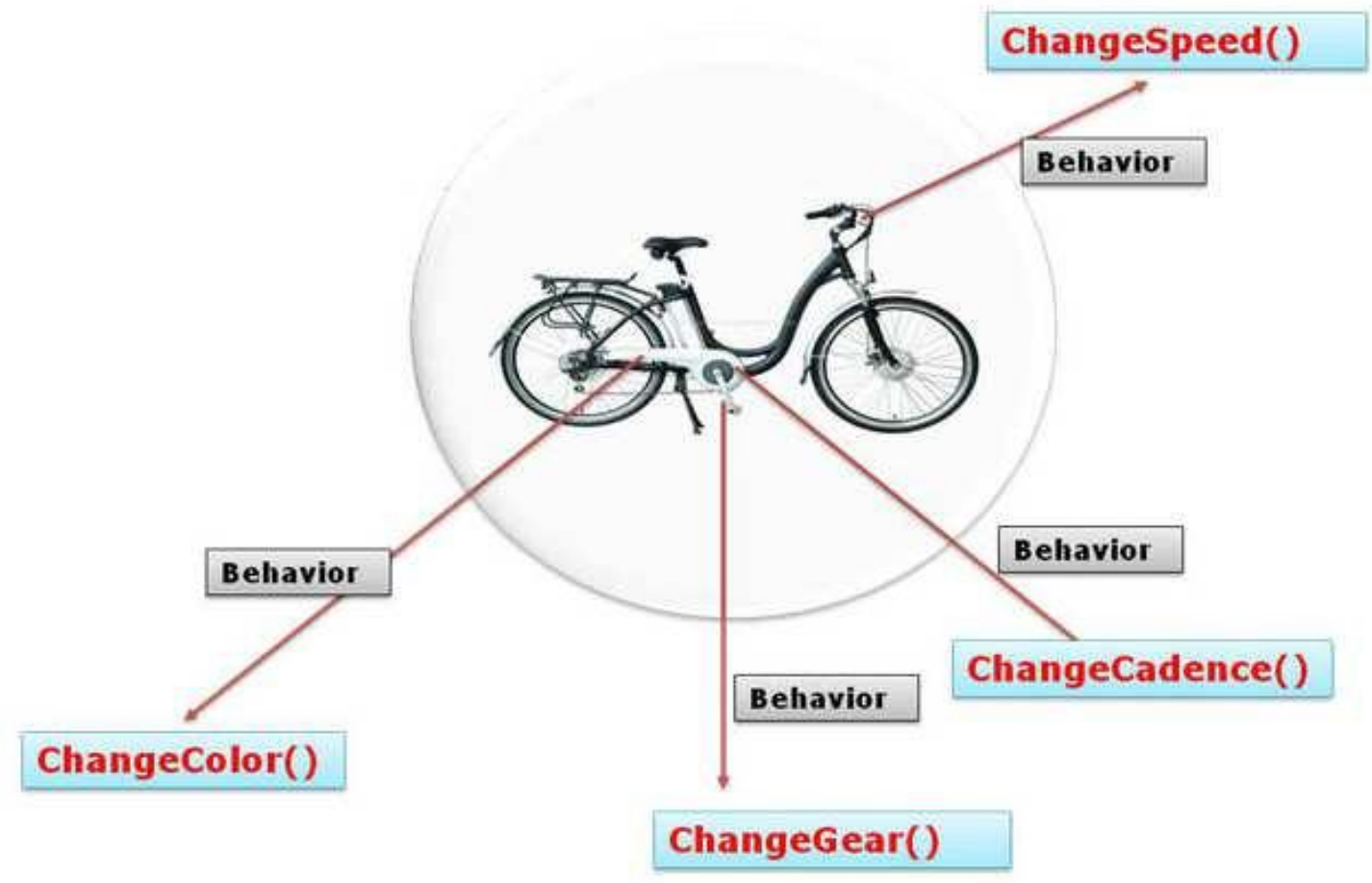
$\Rightarrow$

Attributes



Attributes: variables that describe object characteristics

Methods



Methods: functions that implement the behavior of the object

Theory session 1

General information

Topic 1: Introduction and the concept of objects

Object-oriented programming

Object-oriented programming

Programming:

- "Programming involves tasks such as analysis, generating algorithms, profiling algorithms' accuracy and resource consumption, and the implementation of algorithms [...] in a chosen programming language"
- "The purpose of programming is to find a sequence of instructions that will automate the performance of a task"

Object-oriented programming

Programming:

- "Programming involves tasks such as analysis, generating algorithms, profiling algorithms' accuracy and resource consumption, and the implementation of algorithms [...] in a chosen programming language"
- "The purpose of programming is to find a sequence of instructions that will automate the performance of a task"

Object-oriented programming:

- "In object-oriented programming, computer programs are designed by making them out of objects that interact with one another"

Delegation

- "A mechanism by which an object delegates responsability for a task to another object"
- Fundamental principle in object-oriented programming
- Promotes abstraction and reuse

Example

- John Smith wants to travel from Barcelona to Paris
- Which are the possible solutions to this problem?

Possible solutions

1. John Smith travels walking (only solution without delegation)
2. John Smith travels by car (delegates the task of moving to a car object)
3. John Smith travels by airplane
   - Delegates the task of searching for flights to a travel website
   - Delegates the task of flying to the airline
   - etc.

Discussion

- Different objects interact to reach a solution
- Abstraction: an object does not need to know how another object performs a task, only that it is able to perform it
- Reuse: Once created, an object can be used multiple times
- Once we have delegated a task, the other object is responsible for performing it

Object-oriented design

1. Identify the objects that will participate in the solution
2. If an object is already defined, reuse and/or modify
3. If an object is not defined, create a new definition
4. Determine how the objects interact in the solution

Implementation

- Translate the design to code
- There exist a variety of object-oriented programming languages
- Most are text-based, e.g. Java, C++ and Python
- Encapsulation: the code of an object is inaccessible from the outside

Example: "Hello World"

Python program:

```
def main():
    print( "Hello World!" )
```

Java program:

```
class HelloWorldApp {
    public static void main( String[] args ) {
        System.out.println( "Hello World!" );
    }
}
```

Differences

Programs that are not object-oriented:

- Program: sequence of instructions
- Code is grouped by procedures or functions
- Communication: procedure and function calls

Programs that are object-oriented:

- Program: set of object definitions
- Variables and functions are grouped by object type
- Communication: messages between objects

Summary

- Facilitating programming implies bringing the language closer to our way of thinking
- Humans are hardwired to think in terms of objects
- Object-oriented programming promotes abstraction and reuse
- Program: define objects and specify their interaction