Computer Programming

Object-Oriented Programming II

The Return

Willy Picard
Department of Information Technology
The Poznan University of Economics
<*picard@kti.ae.poznan.pl*>

Agenda

- Lecture Goal(s)
- From Italy to Indonesia
- Interfaces, Classes, and Objects
- Attributes and Methods
- Encapsulation
- Inheritance and Polymorphism
- Conclusions

Lecture Goal(s)

Lectures Overview

Fundamental Concepts

- 1: Introduction
- 2: Basic data structures & Statements
- 3: Object-oriented programming I
- 4: Object-oriented programming II
- 5: Object-oriented programming III
- 6: Complex data structures
- 7: Threads & Exception handling

Today's Goal

To provide programming knowledge about object-oriented (OO) programming

LECTURE GOALS

Refreshments and Peanuts

Interface Definition

An interface defines a set of related functionalities (a behavior)

Class Definition

An class defines the implementation of a set of related functionalities (a behavior)

REFRESHMENTS

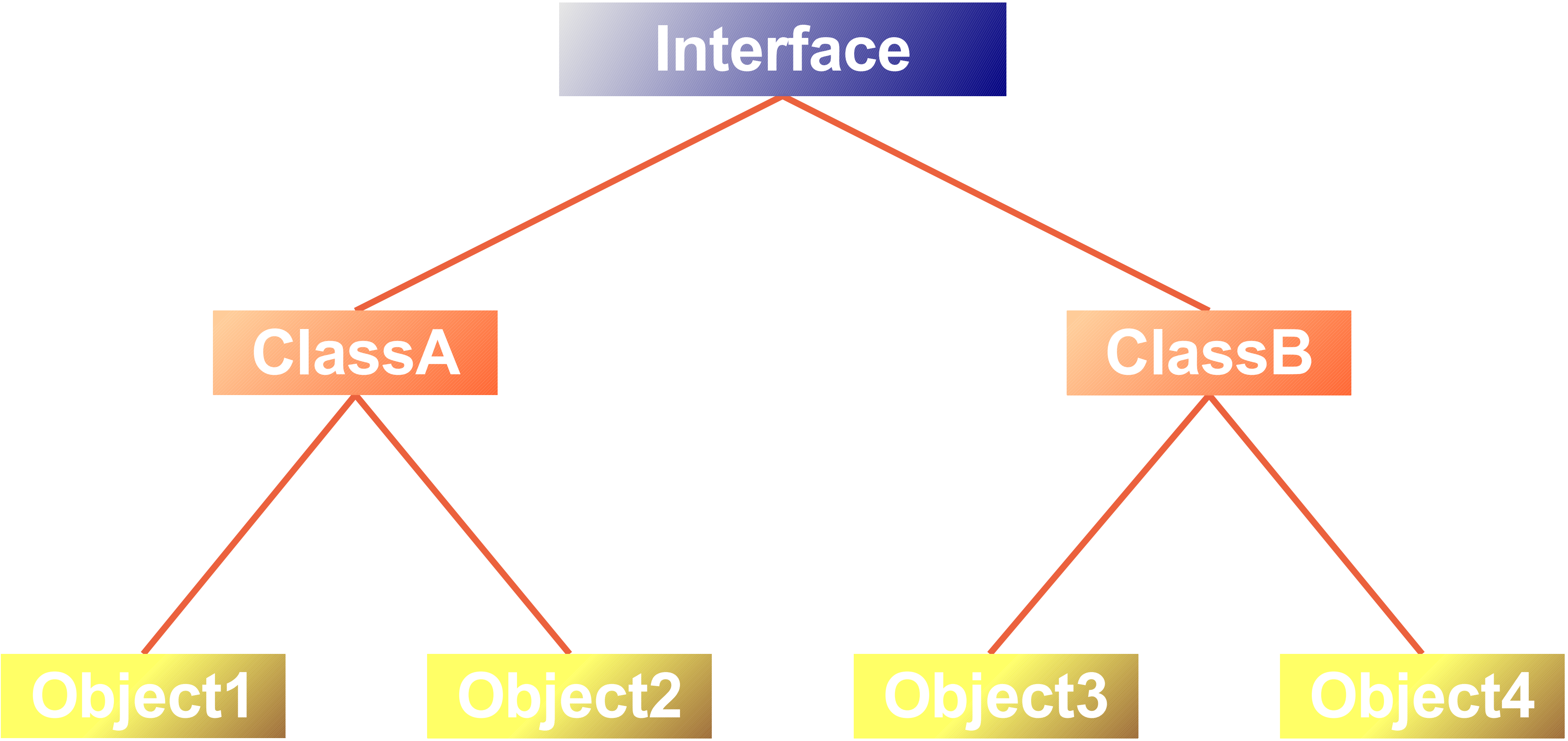
Object Definition

An object is an instance of a class

Classes and Interfaces

A class which implements an interface must define all methods declared in the interface

Summary



Attributes and Methods

Attribute Definition

An attribute is a variable used to capture the state of an object

Fields in Java

- Syntax

```
class <name>{
    ...
    <type> <fieldName>;
    ...
}
```

- Example

```
class CitroenC3 {
    ...
    Pedal _gasPedal;
    Engine _engine;
    Registration _registration;
    ...
}
```

Method Definition

A method is an implementation
of a piece of behavior
(a function inside a class)

Messages and Methods

- Objects communicate
  - Message exchange
  - The “doSomething” metaphore
- Messages
  - Synchronous
  - Asynchronous
- Message sending = method calls

Methods in Java

- Syntax

```
interface <name>{
   ...
   <return_type> <methodName>(<parameters>);
   ...
}
class <name>{
   ...
   <return_type> <methodName>(<parameters>){
      ...
   }
   ...
}
```

- The `void` keyword

```
void printCurrentTime();
```

Methods in Java

- Example

```
class CitroenC3 {
   ...
   int speedUp(int strength){
     _gasPedal.press(strength);
     return _engine.getCurrentSpeed();
   }
   ...
}
```

Abstract Methods in Java

- Incomplete definition
- Example

```
abstract class CitroenC3 {
   ...
   int speedUp(int strength){
     _gasPedal.press(strength);
     return _engine.getCurrentSpeed();
   }
   ...
   abstact break();
}
```

Constructor Definition

A constructor is an special “function” which creates an instance of a given class

Constructors in Java

- Syntax

```
class <className>{
    ...
    <className>(<parameters>){
        ...
    }
    ...
}
```

Constructors in Java

- Example

```
class CitroenC3 {
   ...
   CitroenC3(Pedal gas,
             Engine engine,
             Registration registration){
     _gasPedal = gas;
     _engine = engine;
     _registration = registration;
     setSpeed(0);
   }
   ...
}
```

Creating Objects in Java

- Example

```
...
aCar = new CitroenC3(gas, engine,
                     registration);
...
```

- By default
  - the empty constructor

Destructor Definition

A destructor is an special function which deletes an instance of a given class

Destructors in C++

- Syntax

```
class <className>{
      ...
      ~<className>(<parameters>){
           ...
      }

      ...
}
```

Destructors in Java

- No destructors in Java
- Garbage collector
  - Deletes any non-referenced object
- Memory management
  - JVM responsibility
- `null`
  - e.g. `_engine = null;`

Instance *vs* Class Fields

- Fields
  - attributes
  - methods
- Instance field
  - associated to a single instance
- Class Fields
  - common to all instances of a given class

Fields in Java

- By default
  - instance fields
- Class fields
  - `static` keyword
- Example
  - `static Date _currentDate;`
  - `static Date getCurrentDate();`

Static Example in Java

- Xmas tree lighting set

```
class ILight{
   void screw();
   void unscrew();
}
class Light{
   boolean _isScrewed;
   void screw(){...}
   void unscrew(){...}
   boolean isScrewed(){
     return _isScrewed;
  }
}
```

Static Example in Java

```
class Light{
   boolean _isScrewed;
   static int _unscrewedCounter;
   void screw(){
      _unscrewedCounter --;
      _isScrewed = true;
   }
   void unscrew(){
      _unscrewedCounter ++;
      _isScrewed = false;
   }
}
```

Static Example in Java

```
class Light{
   static int _unscrewedCounter;
   ...
   static boolean doesShine(){
      return _unscrewedCounter == 0;
   }
}
```

Conclusions

Conclusions

- Rule 1
  - Use interfaces
- Rule 2
  - Use interfaces
- Rule 3
  - Use interfaces

C Language vs. OOPLs

Coupling between	*classes*
procedures/functions	
data structures	
Code reuse	*inheritance*
Spread code	*classes,* *inheritance*
Description vs. definition	*interfaces,* *encapsulation*

CONCLUSIONS

Example

```
package pl.poznan.ae.compProg;
import java.util.*;

public class Sorter {
  private List _words;

  public void sort(String[] words){
    _words = Arrays.asList(words);
    Collections.sort(_words);
  }
  public String getSortedWords(){
    String sortedString = "";
    for (int i = 0; i< _words.size(); i++){
      sortedString += _words.get(i);
    }
    return sortedString;
  }
 public static void main(String[] args){
    Sorter sorter = new Sorter();
    sorter.sort(args);
    System.out.println(sorter.getSortedWords());
 }
}
```

See you next week

Object-Oriented Programming III

The Revenge