Computer Programming

Object-Oriented Programming III

The Revenge

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
- Refreshments and Peanuts
- Inheritance
- Polymorphism
- Encapsulation
- 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

Refreshments and Peanuts

Example of Interface

```
class IAnimal{
        int getWeight();
        String getName();
        void shout();
        void eat();
        void eat(int FoodAmount);
}
```

Example of Class

```
class Cat implements IAnimal{
    int _weight;
    String _name;
    Cat(int weight, String name){
        _weight = weight;
        _name = name;
    }
    ...
}
```

```
Cat myCat = new Cat(1200, "Felix");
```

Example of Class

```
class Cat implements IAnimal{
    int _weight;
    String _name;
    int getWeight(){
        return _weight;
    }
    String getName(){
        return _name;
    }
    ...
}
```

Example of Class

```
class Cat implements IAnimal{
        ...
        void shout(){
            System.out.println(“Miaow”);
        }
        void eat(){
            _weight += 200;
        }
        void eat(int foodAmount){
            _weight += foodAmount;
        }
}
```

System.out.prinln(myCat.getName()+ " says "+ myCat.shout());

Java String

- A String is
  - An object
  - A chain of characters
- Example

```
String catName = new String(“Felix”);
String catName = “Felix”;
```

- Useful “tricks”
  - Adding strings: `“My cat's name is “ + catName;`
  - Comparing: `catName.equals(“Felix”);`

The main() method

- `public class Cat{`
  `    public static void main(String[] args){...}`
  `}`
- Run by `java Cat`
- Arguments
  - `String[] args`
  - Number of arguments: `args.length`
- Example
  - `java Cat “Felix” “1200”`
  - `args[0] = “Felix”, args[1] = “1200”`

P E A N U T S

Inheritance

Classes and Interfaces

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

Classes and Interfaces in Java

- Syntax

```
class <className> implements <interfaceName>{
    ...
}
```

- Example

```
class Cat implements IAnimal{
    ...
}
```

Classes and Subclasses

A subclass which extends a class inherits attributes and methods from the its superclass and all its ancestors

Classes and Subclasses in Java

- Syntax

```
class <className> extends <parentClassName>{
    ...
}
```

- Example

```
class PersianCat extends Cat{
    ...
}
```

Overriding Methods

- Redefinition of a method in a subclass
  - specialization
  - possible code reuse
- Identical method signature
- Constructors

Overriding Methods in Java

```
class Cat {
   void eat(){
         _weight +=200;
   }

}

class PersianCat extends Cat {
   boolean _isSleeping = false;

   void eat(){
         super.eat();
         takeANap();
   }

   void takeANap(){ _isSleeping = true;}

}
```

“this.” in Java

```
class PersianCat extends Cat {
    int foodAmount;


    void eat(int foodAmount){
        this.foodAmount += foodAmount;
        super.eat(foodAmount);
        takeANap();
    }
    void eat(){
        eat(200);
    }
}
```

Overriding Constructors in Java

```
class Cat {
   Cat(String name, int weight){
         _name = name;
         _weight = weight;
   }
}

class PersianCat {
   PersianCat(String name, int weight){
         super(name, weight);
         _foodAmount = 0;
         _isSleeping = false;
   }
}
```

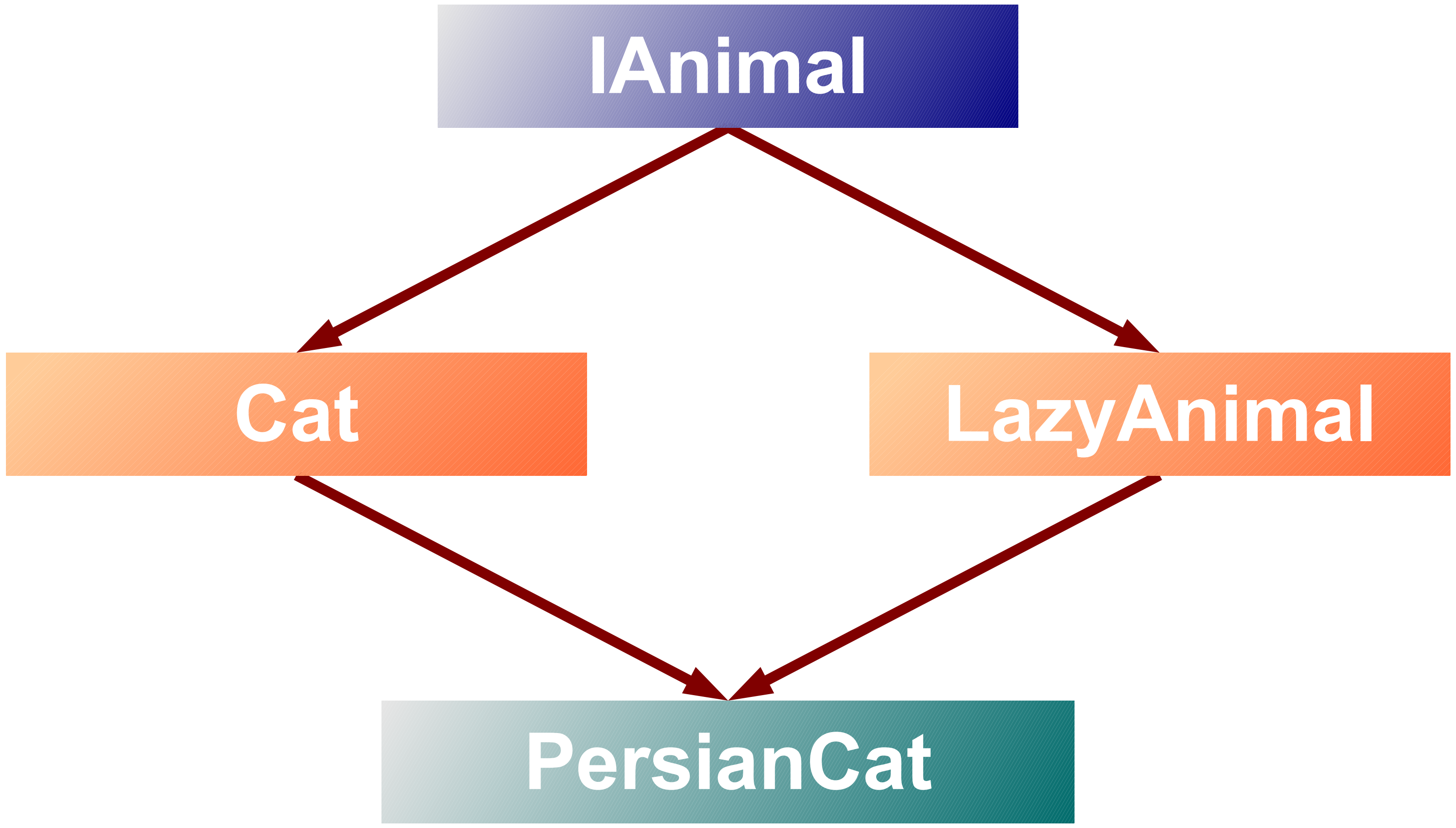
“this()” in Java

```
class PersianCat {

    PersianCat(String name, int weight,
               int foodAmount,
               boolean isSleeping){
        super(name, weight);
        _foodAmount = foodAmount;
        _isSleeping = isSleeping;
    }

    PersianCat(String name, int weight){
        this(name, weight, 0, false);
    }
}
```

Multiple Inheritance

- Interfaces
  - may extends 0, 1 or many interfaces
  - no implementation → no ambiguity
- Classes
  - may extends 0, 1 or many interfaces
  - no implementation → no ambiguity
  - may extends 0, 1 or many classes
  - the “diamond” issue

The “Diamond” Issue

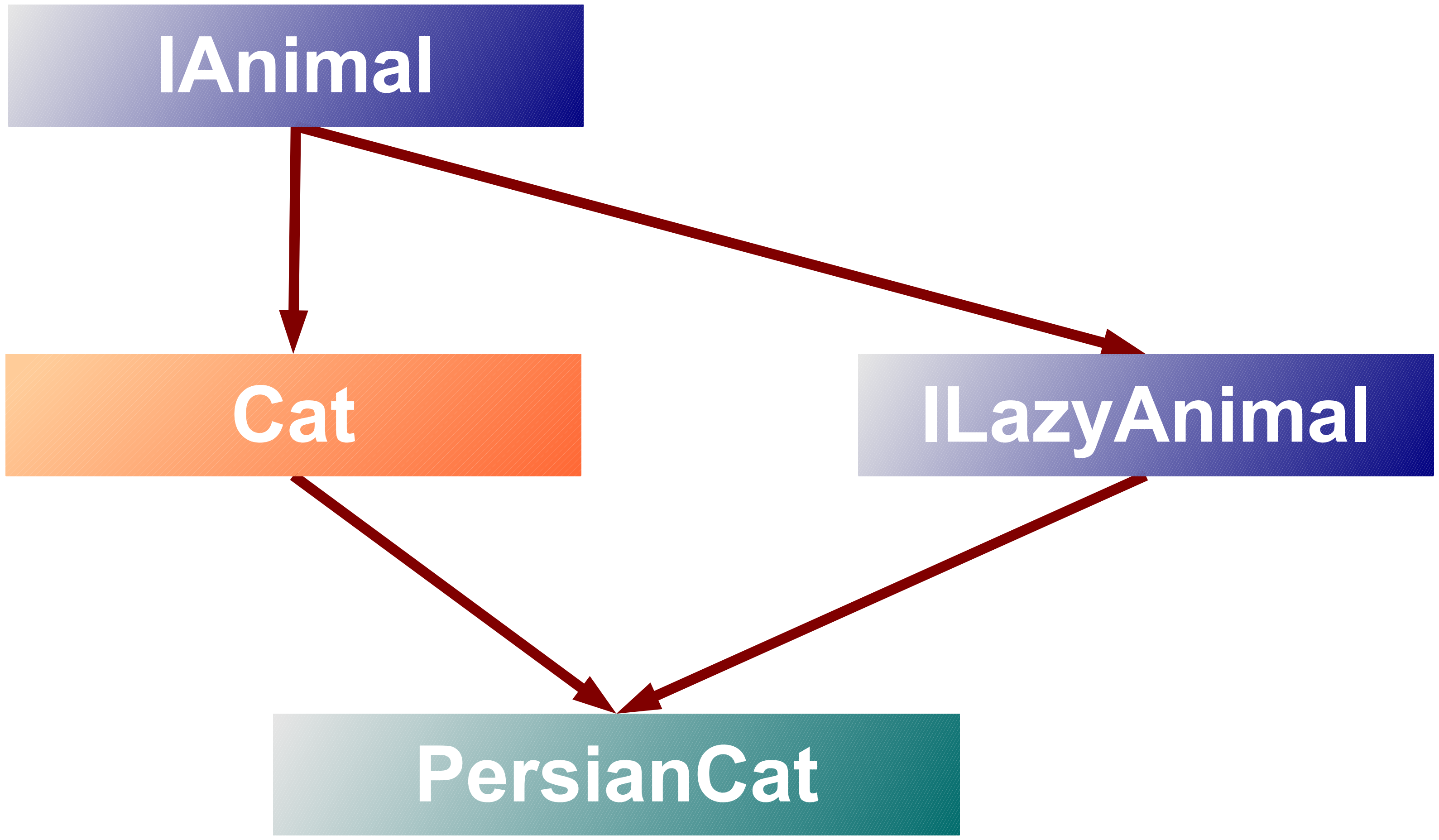


Polymorphism

Polymorphism Definition

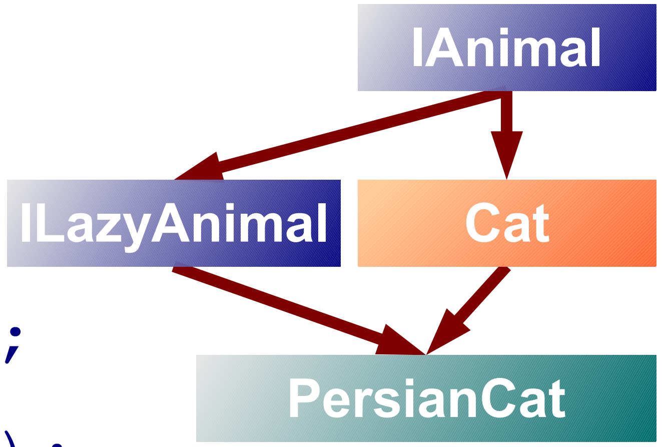
Polymorphism is the ability of some objects to present/deal with various forms (various aspects)

One Object, Various Forms



One Object, Various Forms in Java

- ~~`IAnimal cat = new IAnimal();`~~
- `IAnimal cat = new Cat();`
- `IAnimal cat = new PersianCat();`
- ~~`IAnimal cat = new ILazyAnimal();`~~
- `Cat cat = new Cat();`
- `Cat cat = new PersianCat();`
- `ILazyAnimal cat = new PersianCat();`
- `PersianCat cat = new PersianCat();`



Casting Definition

Casting is the operation of changing the form of a given object

Casting in Java

- `ICat cat1 = new PersianCat();`
- `ILazyAnimal cat2 = new PersianCat();`
- `cat1.takeANap()` is incorrect
- `cat2.takeANap()` is correct

-
  ```
  ILazyAnimal cat3 =
        (ILazyAnimal) cat1;
  ```
- `cat3.takeANap()` is correct

Dealing with Polymorphic Objects

- Overloading methods
  - in one class
  - many methods
  - one method name
  - different parameters
- Overridding methods
  - in many interfaces, classes, subclasses
  - the same method
  - different implementations

Overloading

- Various methods with different arguments
- Checked during compilation
- Example

```
class AnimalOwner {
    void feed(Cat aCat) {...}
    void feed(PersianCat aCat) {...}
 Cat myCat;
 PersianCat aPersianCat;
 ...
 feed(myCat);
 feed(aPersianCat);
```

Overridding

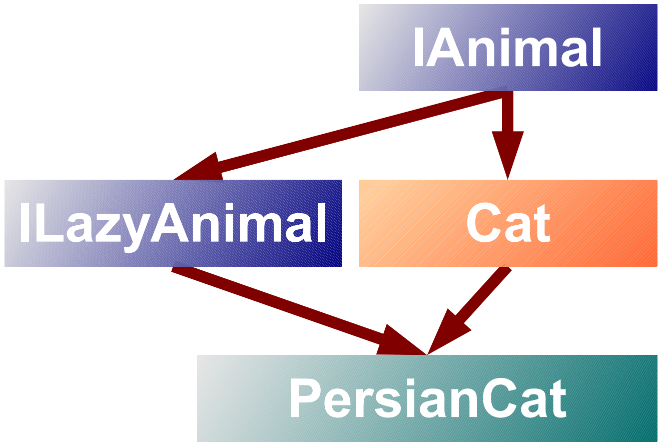
- One method
  - with different implementations
  - in various interfaces, classes and subclasses
- Checked during run-time
- A.k.a
  - Late binding
  - Dynamic binding
  - Run-time binding

Overriding in Java

```
class Cat {
   void eat(){
         _weight +=200;
   }
}
class PersianCat extends Cat {
   boolean _isSleeping = false;
   void eat(){
         super.eat();
         takeANap();
   }
   void takeANap(){ _isSleeping = true;}
}
```

Late Binding in Java

```
IAnimal myCat = new Cat();
IAnimal aPersianCat =
    new PersianCat();


myCat.eat(){
        _weight +=200;
}


aPersianCat.eat(){
        super.eat();
        takeANap();
}
```



Encapsulation

Packages in Java

- Grouping classes
- Package name
  - Empty: default package
  - `(<identifier>.)* <identifier>`
  - e.g. `java.lang`
- Declaration
  - `package <packageName>;`
- Use
  - `import <packageName>.*;`
  - `import <packageName>.<className>;`

Hiding Implementation Details

- Rule 1
  - No field is visible outside the class within it is define
- Rule 2
  - A method is visible iff it is used by another class
- Rule 3
  - Program again interfaces

Visibility in Java

- A set of modifiers
  - `private`
  - `protected`
  - `public`
- Format
  - *<modifier> <field>*
  - e.g. `private int _weight`

Visibility Rules in Java

	Class	Subclass	Package	World
Private	X			
Protected	X	X	X	
Public	X	X	X	X
package (empty)	X		X	

Conclusions

C Language vs. OOPLs

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

Golden Rules

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

Golden Rules

- Rule 4
  - Hide everything that should not be visible
- Rule 5
  - Decouple
- Rule 6
  - Give responsibility to your objects

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