A. Bechini
Course: Java

Introducing Java

Java: Variable Shading, Method Overriding. Polymorphism by Inheritance

Introducing Java

Maintenance and Code Reusability

- Inheritance increases the code reusability because part of the code in the base class is still valid for the derived classes, and thus it's not necessary to rewrite it again.
- Code maintenance is often simplified: some changes required in several classes can be put in the higher levels of the hierarchy, and they are automatically propagated "downwards."
- E.g.: if we must consider the social security number for different categories of people (in different classes), we can insert it directly in their common "root" class (Person)

Introducing Java

Re-definition of Methods (Overriding)

We have "method overloading" whenever a sub-class defines a method already present in a super-class

An overriding method must have the same name and the same signature of the overridden one. Otherwise, if only the names are equal, it becomes a case of overloading

The overriding method hides the corresponding one in the superclass.

```
class A {
  void methodX(){
  System.out.println("Class A");
  }
}
```

```
class B extends A {
  void methodX(){
  System.out.println("Class B");
  }
}
```

```
B objB=new B();
objB.methodX();
```

output → Class B

Introducing Java

Re-definition of Methods (Overriding)

```
class Fruit {
  void peel(){
  System.out.println("Class Fruit");
  }
}
```

```
class Peach extends Fruit {
  void peel(){
  System.out.println("Class Peach");
  }
}
```

```
class Example {
  public static void main(String argv[]){
    Fruit oneFruit=new Fruit();
    Peach onePeach=new Peach();
    oneFruit.peel();
    onePeach.peel();
    oneFruit=onePeach;
    oneFruit.peel();
  }
}
```

The first time the peel() method is invoked on the object oneFruit, the version defined in class Fruit is run

The second time such a method is invoked on the object oneFruit, it actually refers to a Peach, thus the Peach version of the peel() method is run.

This is a particular polymorphic behavior.

Introducing Java

The *this* keyword

The this keyword may take different meanings:

1 Reference to the current object

- t=this.x This instruction assigns t the value of the variable x in the current object
- this.methodA(this) Call methodA of the current object, and it passes a reference to the current object as an argument
- return this It returns a reference to the current object

Introducing Java

The *this* keyword

Often this can be omitted.

Instance variables (I.e. non-static) and methods of the current class can be invoked without any this:

t=this.x and this.methodA(this)

can be rewritten as

t=x and methodA(this)

The keyword this must be explicitly inserted whenever some local variables hold the same name of instance variables:

```
class Example {
  int x;
  int methodA(int x) {
    return x;
  }
}
```

The method parameter is returned

The instance variable is returned

```
class Example {
  int x;
  int methodA(int x) {
    return this.x;
  }
}
```

A. Bechini
Course: Java

The *this* keyword

"This" can be used only in instance methods, as it is a reference to the current instance of the class

Class methods (I.e. static) cannot use "this"

2 The keyword this can be used as a "method name," in order to invoke a constructor

this(arg1, arg2, …)

```
class Rect {
  int x1, y1,  x2, y2;

  Rect(int x2, int y2) {
      this.x2=x2;
      this.y2=y2;
  }

  Rect(int a, int b, int c, int d) {
      this(c,d);
      x1=a;
      y1=b;
  }
}
```


The keyword *super* (1)

The keyword super is used like the keyword this and it gives us the chance to refer to variables and methods in the superclass (costructors included)

```
class Vehicle {

  String model;
  String color;

  Vehicle(String mod, String col) {
      model=mod;
      color=col;
  }
}
```

```
class MotorVehicle extends Vehicle {

  int fuelLevel;

  void refill() {
    if(fuelLevel==100)
            System.out.println("Already Full");
    else {
      fuelLevel=100;
      System.out.println("Refilled!");
    }
  }

  MotorVehicle(String mod, String col) {
     super(mod, col);
  }
}
```

Invocation of the constructor in the superclass

The keyword *super* (2)

```
class A {

  void methodX(){
  System.out.println("Class A");
  }
}
```

```
class B extends A {

  void methodX(){
    System.out.println("Class B");
  }

  void methodY(){
     methodX();
     this.methodX();
     super.methodX();
  }
}
```

Through super it's possible to access to variables and methods in the superclass that have been re-defined in the current class

Esercizi

- Creare la classe Persona con alcuni attributi e metodi
  - fisici(statura, …), numeroCartaId, indirizzo, …
  - getNumCartaId(), …
- Derivarne due sottoclassi: Studente e Professore ognuna con altri attributi e metodi
  - anno_di_corso, materia_insegnata, scuola,
  - getScuola(), …
- Fornire tutte le classi di costruttori e di un metodo che ne stampi lo stato (il valore di tutte le variabili)
- Creare un array di Persona e inserire al suo interno qualche oggetto Studente e qualche oggetto Professore e qualche oggetto Persona
- Stampare il contenuto dell'array