Patrick GIRARD
University of Poitiers
2009-2014©

Object-Oriented Programming
in Java^{tm}

Basics of Inheritance

Chapter 5 - Section 2

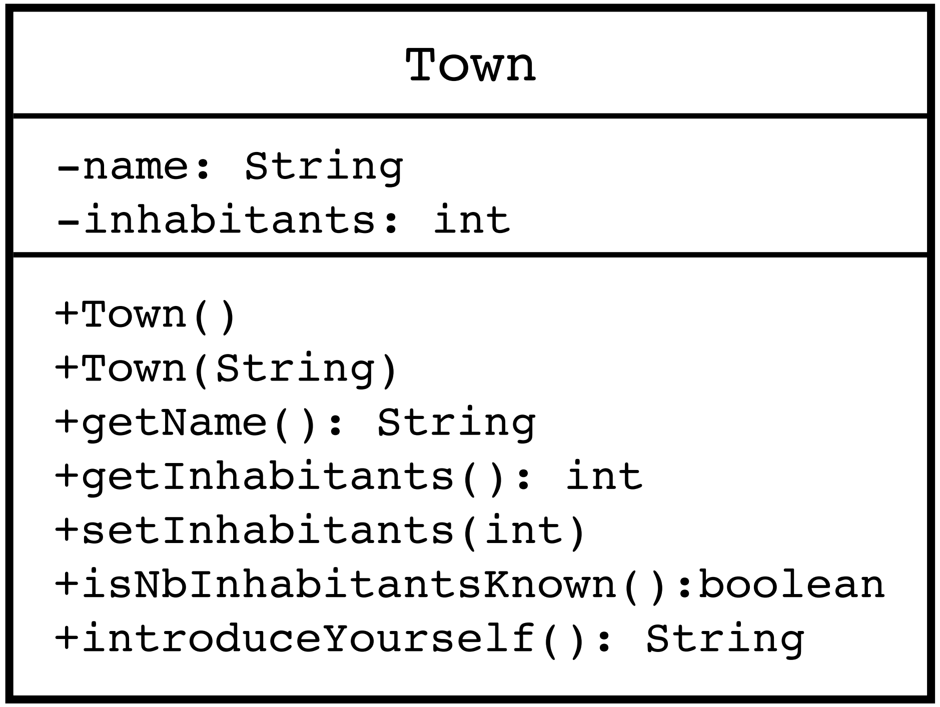
Table of contents

Simple inheritance Another example

- The Town concept
  - We take into account the name and the number of inhabitants
  - The name cannot be changed, nor be null
  - The name of inhabitants may be unknown
  - A Town object is able to describe itself in a String

Simple inheritance
Another example

- The Town class



```
Town
-name: String
-inhabitants: int
+Town()
+Town(String)
+getName(): String
+getInhabitants(): int
+setInhabitants(int)
+isNbInhabitantsKnown():boolean
+introduceYourself(): String
```

Simple inheritance

Another example

```
public class Town {
   private String name;
   private int inhabitants;
   public Town() { name = "PARIS";}
   public Town(String theName) { name = theName;}
   public String getName() { return name; }
   public int getInhabitants() {
      if (inhabitants<=0) System.exit(0); return inhabitants; }
   public void setInhabitants(int nbInhabitants) {
      if (nbInhabitants>=0) inhabitants = nbInhabitants;
      else System.exit(0); }
   public boolean isNbInhabitantsKnown() {
      return inhabitants > 0; }
   public String introduceYourself() {
      String s = "I am (" + name + ", ";
      if (inhabitants > 0) s += inhabitants + ")"
      else s += « unknown number of inhabitants"
      return s;
   }
}
```

Simple inheritance
Another example

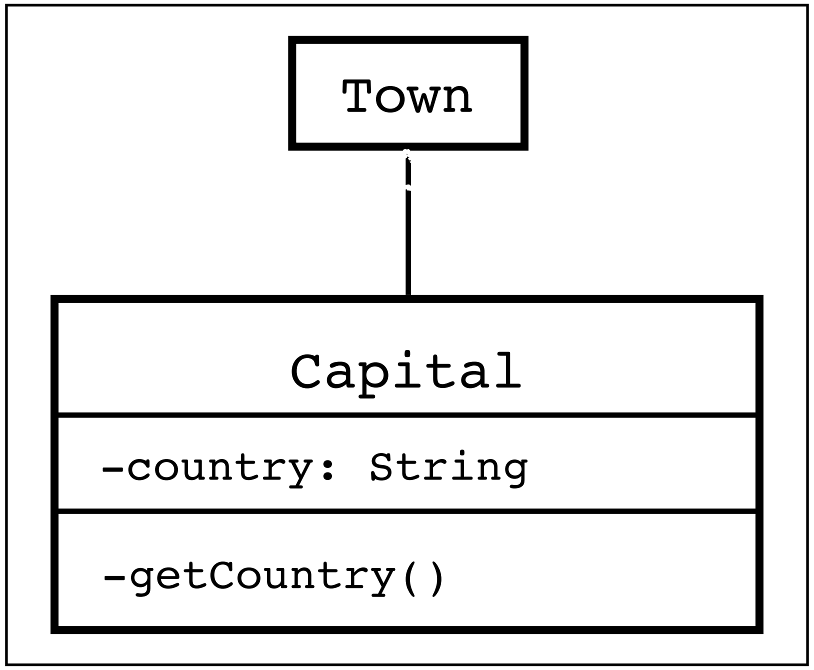
Execution

```
public class Example1 {
   public static void main (String args[]){
      Town t1 = new Town("Oran");
      t1.setInhabitants(800000);
      System.out.println(t1.introduceYourself());
      Town t2 = new Town();
      System.out.println(t2.introduceYourself());
   }
}
```

```
I am (ORAN, 800000)
I am (PARIS, unknown number of inhabitants)
```

Simple inheritance
Another example

- A capital:
  - A town...
  - which is capital of a country



Simple inheritance
Another example

The Capital class: an extension

```
public class Capital extends Town {
   private String country;
   public Capital (String theCountry) {
      country = theCountry.toUpperCase();
   }
   public getCountry() { return country; }
}
```

```
public class Example2 {
   public static void main (String args[]){
      Capital c = new Capital("France");
      System.out.println(c.introduceYourself());
   }
}
```

```
I am (PARIS, unknown number of inhabitants)
```

Digression: the constructors

- We want a class that delivers incremented int

```
public class Number {
   private int num;
   public int newNumber () { return ++num; }
}
```

```
public class Example3 {
   public static void main (String args[]){
      Number incr = new Number();
      for (int i=0; i<5; i++)
         System.out.println(incr.newNumber() + ":");
   }
}
```

```
1:2:3:4:5:
```

Digression: the constructors

- Improvement: choosing the starting num

```
public class Number {
   private int num;
   public Number (int start) { num : start };
   public int newNumber () { return ++num; }
}
```

```
public class Example3 {
   public static void main (String args[]){
      Number incr = new Number(5);
      for (int i=0; i<5; i++)
         System.out.println(incr.newNumber() + ":");
   }
}
```

```
6:7:8:9:10
```

Digression: the constructors

- if we re-run Exemple3.class

```
public class Example3 {
   public static void main (String args[]){
      Number incr = new Number();
      for (int i=0; i<5; i++)
         System.out.println(incr.newNumber() + ":");
   }
}
```

???

```
Number: method <init>()V not found
        at Exemple3.main (Example3.java:11);
```

Digression: the constructors

- Try to recompile Exemple3.java ?

```
public class Number {
   private int num;
   public Number (int start) { num : start };
   public int newNumber () { return ++num; }
}
```

```
public class Example3 {
   public static void main (String args[]){
      Number incr = new Number();
      for (int i=0; i<5; i++)
         System.out.println(incr.newNumber() + ":");
   }
}
```

Number(int) in Number cannot be applied to ()

Digression: the constructors

- Summary
  - A constructor is created by default in each class
  - Any explicitly built constructor suppress the default constructor
  - A parameterless constructor can/must be explicitly defined if needed

Back to our example

- A more convincing constructor: suppress the stupid default constructor

```
public class Town {
   private String name;
   private int inhabitants;
   ~~public Town() { name = "PARIS";}~~
   public Town(String theName) { name = theName;}
   …
   }
}
```

Back to our example

- Yes, but... damnit !

```
public class Capital extends Town {
   private String country;
   public Capital (String theCountry) {
      country = theCountry.toUpperCase();
   }
   public getCountry() { return country; }
}
```

Town(String) in Town cannot be applied to ()

Back to our example

First try: direct access to the name?

```
public class Capital extends Town {
   private String country;
   public Capital (String theName, String theCountry) {
      name = theName;
      country = theCountry.toUpperCase();
   } …
}
```

name has private access in Town

Back to our example

- Using the protected keyword

```
public class Town {
   **protected** String name;
   private int inhabitants;
     …
```

Bad solution

```
public class Capital extends Town {
   private String country;
   public Capital (String theName, String theCountry) {
      name = theName;
      country = theCountry.toUpperCase();
   } …
}
```

Back to our example

- The danger of the protected keyword

```
public class Town {
   protected String name;
   private int inhabitants;
     …
```

Encapsulation breaking

```
public class Freetown extends Town {
   public void setName (String theName) {
      name = "Hacked by me! » + theName;
   } …
}
```

Back to our example

- More convincing: an explicit constructor call ?

```
public class Capital extends Town {
   private String country;
   public Capital (String theName, String theCountry) {
      Town(theName);
      country = theCountry.toUpperCase();
   } …
}
```

cannot resolve symbol: method Town(String)

Good idea - wrong solution

Back to our example

- The right solution: the super keyword

```
public class Capital extends Town {
   private String country;
   public Capital (String theName, String theCountry) {
      super(theName);
      this.country = theCountry.toUpperCase();
   } …
}
```

Ok !!!

What is "this" ?

Digression: the this keyword

- We already used this, the current object: this.name = name

```
public class Town {
   private String name;
   private int inhabitants;
   public Town(String name) {
      this.name = name;
   }
   public Town(String name, int nbInH) {
      this(name);
      setInhabitants(nbInH);
   }
   …
}
```

Back to our example

- Summary
  - Class extension
    - Adding attributes and/or methods
  - Redefining methods
    - The constructor (obligatory)
    - The introduceYourself() function