Patrick GIRARD
University of Poitiers

Writing Classes and instanciating Objects in Java with BlueJ

Chapter 2 - Section 4

Table of contents

Encapsulation

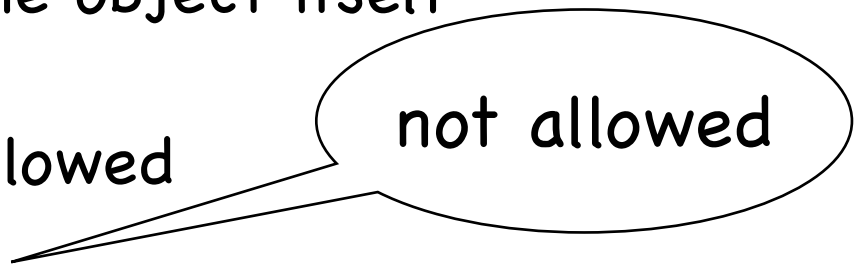
- With encapsulation, object can:
  - manage their state
  - protect them
  - control their usage
- Also called "Data Hiding"

Tutorial: the class "Person"

- Requirements :
  - a name and a working company
    - the name always must be present
    - the name cannot change
    - the working company is optional, and may change
    - name and working company must always be written in UPPER CASE LETTERS
  - a Person knows how to introduce him/herself

First step: protecting

- Rule 3: the name cannot change
  - The private status allows to prevent modifying an attribute from outside the object itself
    - private String name;
  - External access are disallowed
    - aPerson.name = ...
  - Only internal access are allowed
    - from inside internal methods...



The "private" modifier

```
class Person {
  private String name;
  private String company;
  public void whoAmI(){… }
}
class Test {
public static void main(String args[]) {
  Person aPerson;
  aPerson = new Person();
  aPerson.name = "JONES"; // error
  aPerson.company = "SUN"; // error
  aPerson.whoAmI();
}
}
```

Step 2: controlling access

- Rule 5: name and company in upper case letter
  - Only member functions can access private objects
- Accessors
  - to access the value
- Mutators
  - to change the value

Accessor and Modifier

```
class Person {
  private String name;
  private String company;
  public String getCompany(){… };
  public void setCompany(String nameComp){… };
}
class Test {
public static void main(String args[]) {
  Person aPerson;
  aPerson = new Person();
  aPerson.setCompany ("SUN");
  System.out.println(aPerson.getCompany());
}
```

Implementation

```
class Person {
  private String name;
  private String company;
  public String getCompany(){
    return company;
  };
  public void setCompany(String nameComp){
    company = nameComp.toUpperCase();
  };
}
```

Step 3: initializing data

- Rule 2: name always exists
  - even just after creation
- The wrong solution
  - create the object
  - set the name

```
class Person {
  private String name;
  private String company;
  public void setName(String n) {name = n};
}
class Test {
public static void main(String args[]) {
  Person aPerson;
  aPerson = new Person();
  aPerson.setName("JONES"); }
}
```

Why wrong ?

Step 3: initializing data

- The good solution
  - Defining a constructor

```
class Person {
  private String name;
  private String company;
  public Person (String theName) {
    name = theName.toUpperCase();
    company = "?";
  }
}
class Test {
  Person aPerson = new Person("JONES");
  ...
```

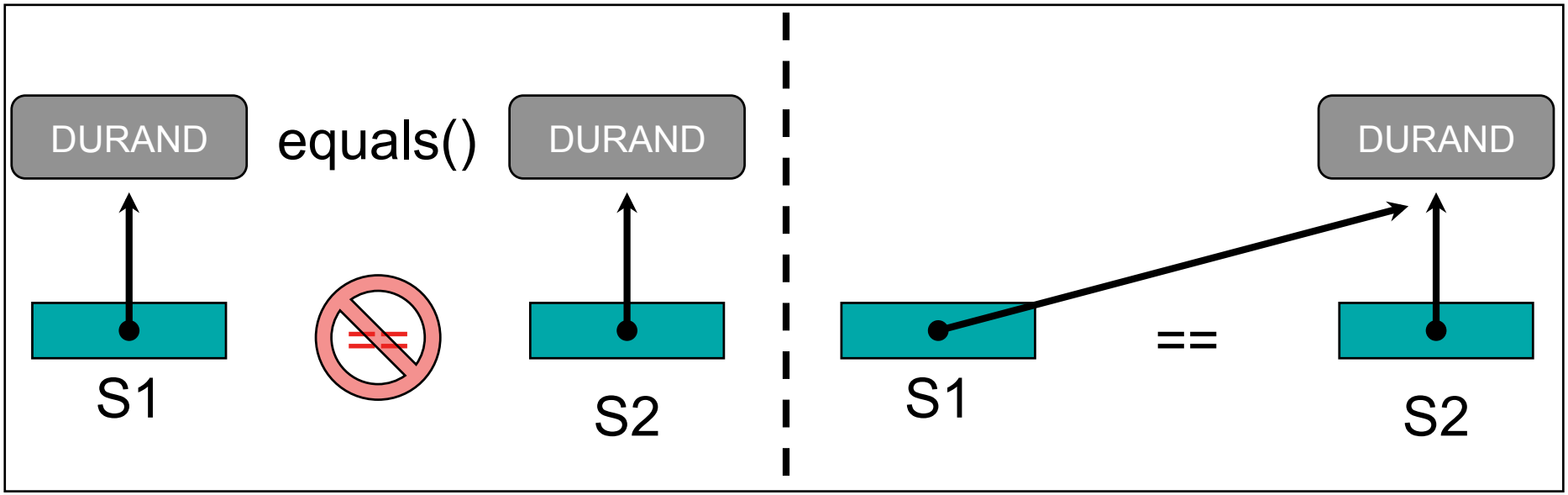
Why right ?

More implementation

```
class Person {
  private String name;
  private String company;
  …
  public void whoAmI () {
    System.out.println("My name is " + nom);
    if (societe.equals("?"))
      System.out.println("I'm unemployed");
    else
      System.out.println("I work at " + company);
  }
}
```

More tips

- Comparing objects ?
  - if (company.equals("?"))



More on constructors

- More than one constructor

```
class Person {
  private String name;
  private String company;
  public Person (String theName) {… }
  public Person (String theName, String theCompany) {
    name = theName.toUpperCase();
    company = theCompany.toUpperCase();
  }
}
```

Improving our software

- "No company" is coded by '?', but...

```
class Person {
  public Person (String theName) {
    name = theName.toUpperCase();
    company = "?";
  }
}
```

- more natural... new task-oriented methods

  losing your job : "loseMyJob()"
  getting a job : "beHired()"

Improving our software

- Company names are not protected
  - '?' can be entered
  - Names can be given without upper case letters
- The solution
  - Company name verification
    - validateCompany();

Private !

Solution

```
class Person {
  …
  private static final String noCompany = "?";
  private String validateCompany(String nameCompany) {
    if (nameCompany.equals(noCompany) ||
        nameCompany.length() > 32 ) {
      System.out.println ("Company name incorrect");
      System.exit(1); //end of program
    }
    return nameCompany;
  }
}
```

Solution

```
class Person {
  …
  void loseMyJob () {
    if (company.equals(noCompany)) {
      System.out.println("Already unemployed!");
      System.exit(2);
    }
    company = noCompany ;
  }
  void beHired (String theCompany) {
    if ( ! company.equals(noCompany)) {
      System.out.println("Already employed!");
      System.exit(3);
    }
    company = validateCompany(theCompany) ;
  }
}
```

Answers (1)

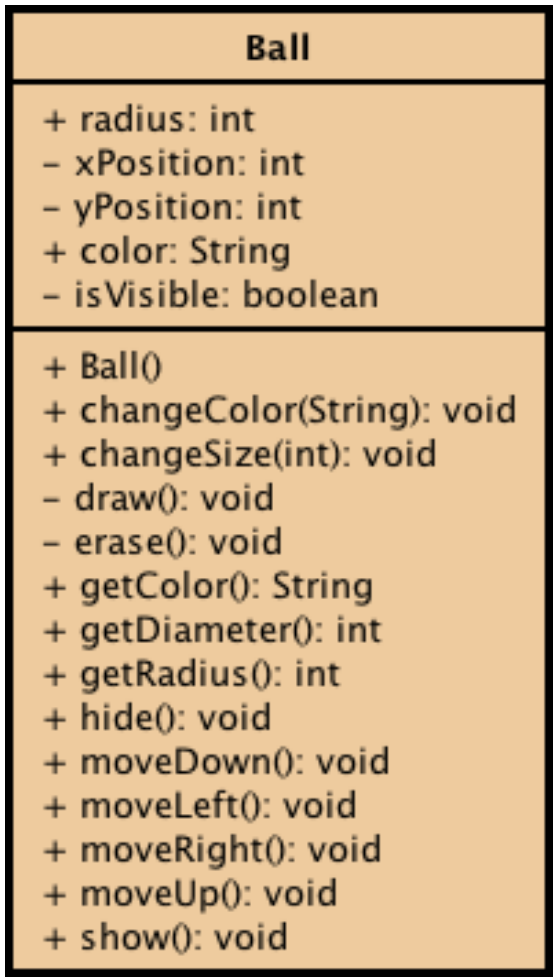
- Private/Public Attributes
  - Attributes should always be private, to avoid bad usage
  - Be careful to create all needed public methods to manage the private attributes
  - Do not forget initializing attributes in the constructor

Answers (2)

- Calculated attributes
  - Avoid duplication of values by using calculated attributes
- Hidden attributes
  - Hidden attributes are generally associated to states. Be careful identifying states.
- Public/Private Methods
  - Methods are mainly public
  - Use private methods to avoid code duplication

Back to the Ball example

- Obvious radius and color attributes have public state
- Why is it a bad solution?



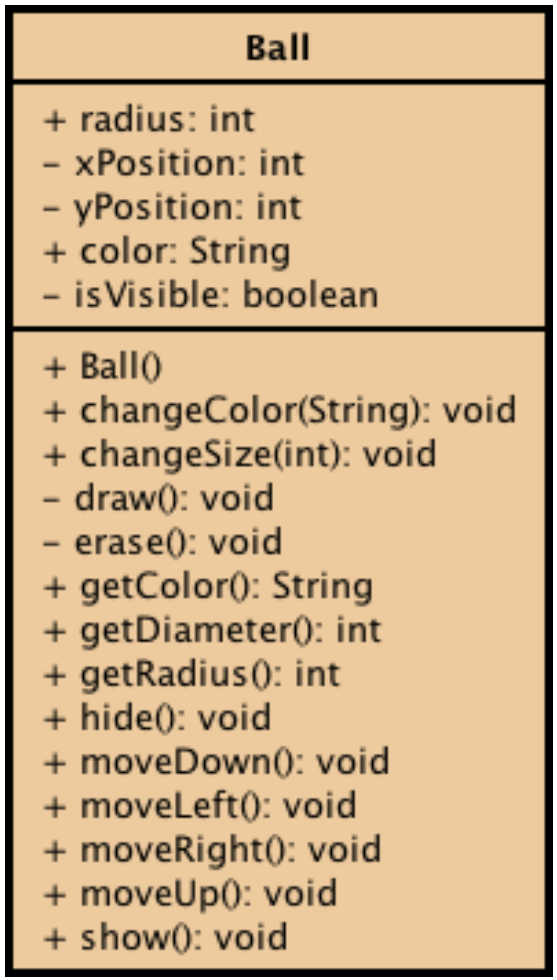
Answer (1)

```
/**
 * Change the size to the new size (in pixels). Size must be >= 0.
 */
public void changeSize(int newRadius)
{
    erase();
    radius = newRadius;
    draw();
}

/**
 * Change the color. Valid colors are "red", "yellow", "blue", "green",
 * "magenta" and "black".
 */
public void changeColor(String newColor)
{
    color = newColor;
    draw();
}
```

Back to the Ball example

- xPosition and yPosition attributes are private
- Why is it a good solution?



Answer (1)

```
/**
 * Move the circle a few pixels to the right.
 */
public void moveRight()
{
    xPosition += 20;
    draw();
}

/**
 * Move the circle a few pixels to the left.
 */
public void moveLeft()
{
    xPosition-= 20;
    draw();
}
```

Tips

- Respect a common general organisation
  - Attributes
  - Constructors
  - Methods
- Respect the Java naming conventions
  - Accessor -> getX, isX
  - Modifiers -> setX, changeX