More on Classes

Encapsulation

Produced by: Dr. Siobhán Drohan
Mairead Meagher

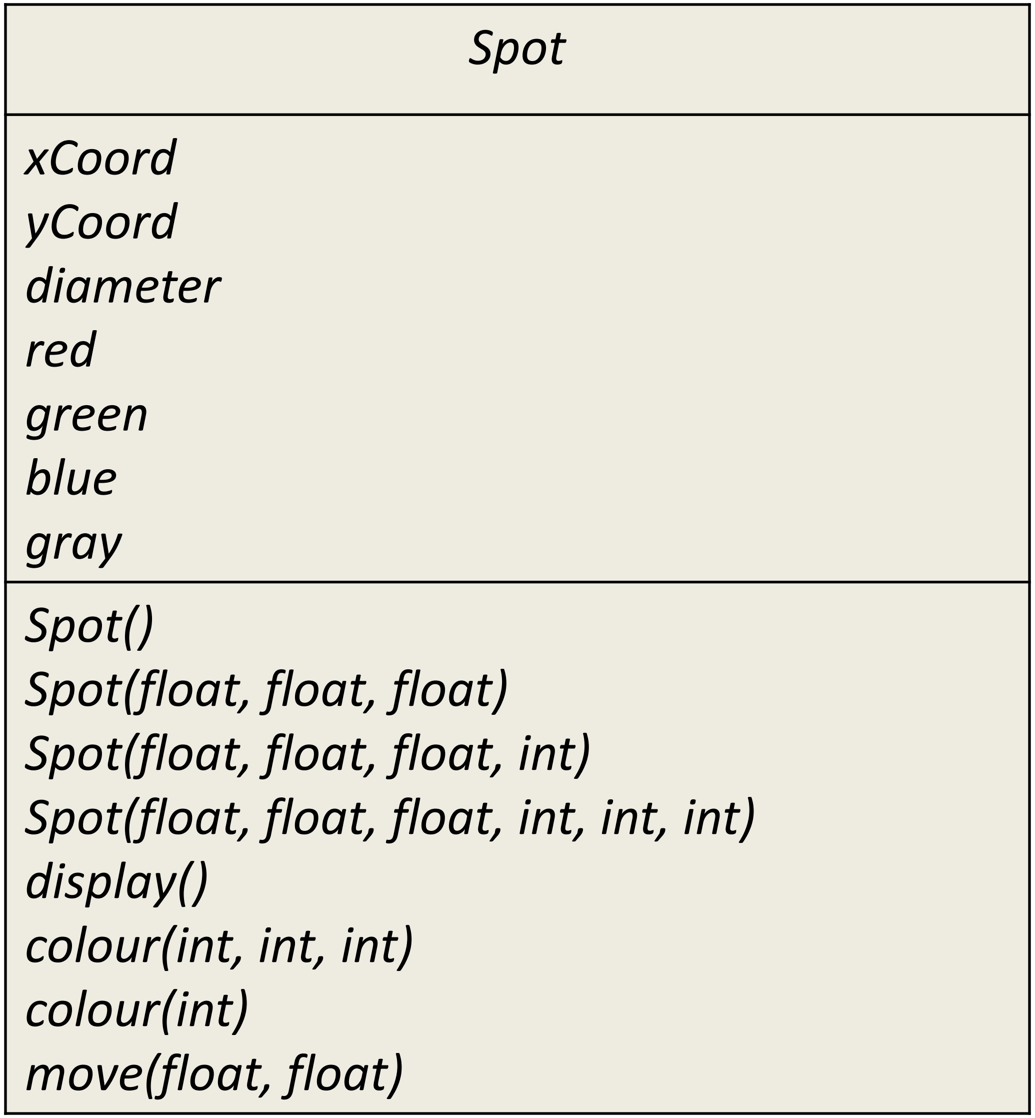
Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Department of Computing and Mathematics
http://www.wit.ie/

Topics list

- Recap: Version 6.1
- Design smells!
- Encapsulation
- Refactoring Spot:
  - Access Modifiers
  - Accessors and Mutators
  - Validation
- Game of Pong
  - Ball class

Class Diagram for Spot Version 6.1

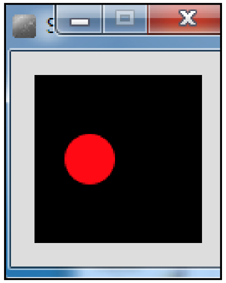


Spot

xCoord
yCoord
diameter
red
green
blue
gray

Spot()
Spot(float, float, float)
Spot(float, float, float, int)
Spot(float, float, float, int, int, int)
display()
colour(int, int, int)
colour(int)
move(float, float)

Spot Class – Version 6.1



```
class Spot{
  float xCoord, yCoord;
  float diameter;
  int red, green, blue;

 Spot()
  {
  }

 Spot(float xCoord, float yCoord, float diameter)
 {
   this.xCoord = xCoord;
   this.yCoord = yCoord;
   this.diameter = diameter;
  }

  // colour methods…
  // display method…
  // move method…
}
```

Source: Reas & Fry (2014)

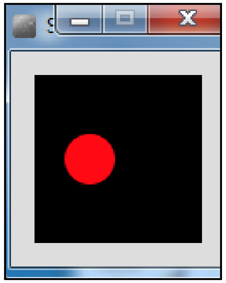
Spot Class – Version 6.1

```
class Spot{
// fields and constructors…

 void display()
  {
    ellipse(xCoord, yCoord, diameter, diameter);
  }

 void colour(int red, int green, int blue)
  {
    this.red = red;
    this.green = green;
    this.blue = blue;
    fill (red, green, blue);
  }

  void colour(int gray){
    this.gray = gray;
    fill (this.gray);
  }
}
```



Source: Reas & Fry (2014)

Spot Class – Version 7.0

```
Spot sp;

void setup()
{
  size (100,100);
  noStroke();
  sp = new Spot(33, 50, 30);
}

void draw()
{
  background(0);
  sp.colour(255, 0, 0);
  sp.diameter = 30000;
  sp.display();
}
```

```
class Spot{
  float xCoord, yCoord;
  float diameter;
  int red, green, blue;

  // constructors…
 void display(){
   ellipse(xCoord, yCoord, diameter, diameter);
  }

 void colour(int red, int green, int blue)
  {
   this.red = red;
   this.green = green;
   this.blue = blue;
   fill (red, green, blue);
  }
 // move methods…
}
```

Source: Reas & Fry (2014)

Our Design Smells!

- We can directly access the diameter field (and all other fields) in the Spot class from another class and set it to a value that is completely preposterous!

- Also, when we directly access a field in a class, we are applying a “behaviour” to that field e.g. resizing the circle:
  - But, aren’t methods supposed to be the “behaviour” for a class???????

Our Design Smells!

- Our design violates one of the basic principles of object-oriented design:

Encapsulation!

Encapsulation

- Encapsulation (data hiding) is a fundamental OOP concept.
- How to achieve encapsulation:
  1. *wrap* the data (fields) and code acting on the data (methods) together as single unit.
  2. *hide* the fields from other classes.
  3. *access* the fields only through the methods of their current class.

http://www.tutorialspoint.com/java/java_encapsulation.htm

Encapsulation in Java

Encapsulation Step	Approach in Java

1. Wrap the data (fields) and code acting on the data (methods) together as single unit.	`public class ClassName`
	`{`
	`    Fields`
	`    Constructors`
	`    Methods`
	`}`

2. Hide the fields from other classes.	Declare the fields of a class as private.
3. Access the fields only through the methods of their current class.	Provide public setter and getter methods to modify and view the fields values.

http://www.tutorialspoint.com/java/java_encapsulation.htm

Refactoring Spot: Access Modifiers

- Java provides a number of access modifiers to set access levels for classes, fields, methods and constructors.
- The four access levels are:
  - Visible to the package, the default. No modifiers needed.
  - Visible to the class only (private).
  - Visible to the world (public).
  - Visible to the package and all subclasses (protected).

http://www.tutorialspoint.com/java/java_modifier_types.htm

Refactoring Spot 7.0: Access Modifiers

```
public class Spot{
  float xCoord, yCoord;
  float diameter;
  int red, green, blue;

  Spot()
  {
  }

  // other constructor
  void display(){
     ellipse(xCoord, yCoord, diameter, diameter);
  }
  // move method…
  // colour methods…
}
```

Encapsulation step 1 is complete; all fields, constructors and methods are all in a single unit, called Spot.

The default access level is package, so our class, methods and fields are all package level access:
→this breaks Encapsulation step 2 i.e. the fields of a class should be private.

Refactoring Spot 7.0: Access Modifiers

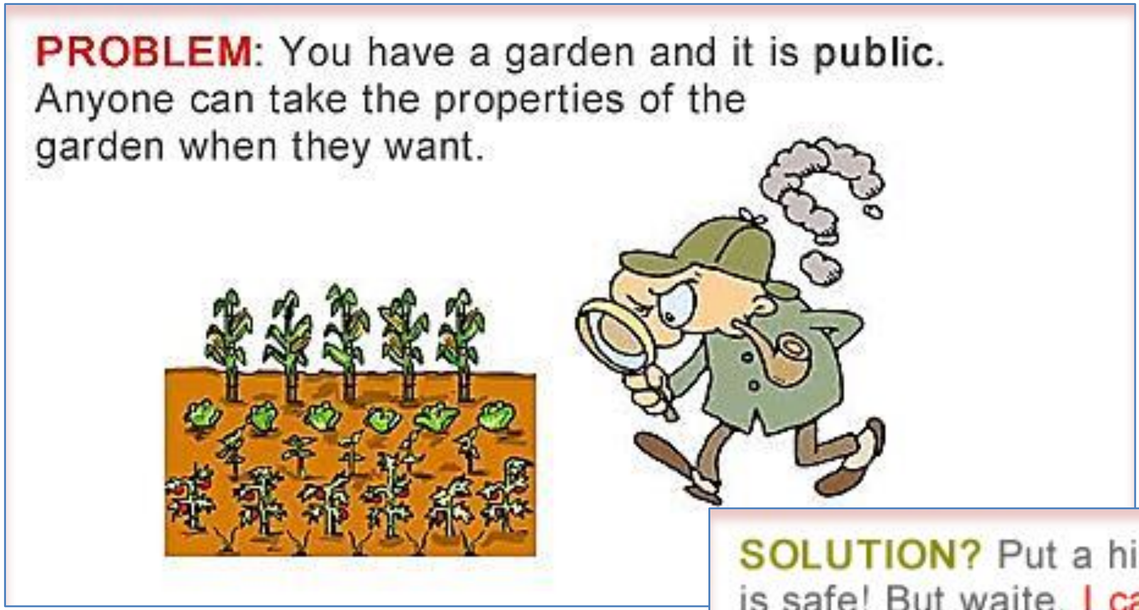
```
public class Spot{
   private float xCoord, yCoord;
   private float diameter;
   private int red, green, blue;

   Spot()
   {
   }

   // other constructor
   void display(){
      ellipse(xCoord, yCoord, diameter, diameter);
   }
   // move method…
   // colour methods…
}
```

To fix Encapsulation step 2, we declare all the fields with private access.

Refactoring Spot 7.0: Access Modifiers



PROBLEM: You have a garden and it is **public**. Anyone can take the properties of the garden when they want.

SOLUTION? Put a high fence around my garden, now it is safe! But waite, I can no longer access my own garden.

http://www.evinw.com/w/

Refactoring Spot 7.0: Access Modifiers

```
public class Spot{
   private float xCoord, yCoord;
   private float diameter;
   private int red, green, blue;
   //constructors…
   //display method…
   // move method…
   // colour methods…
}
```

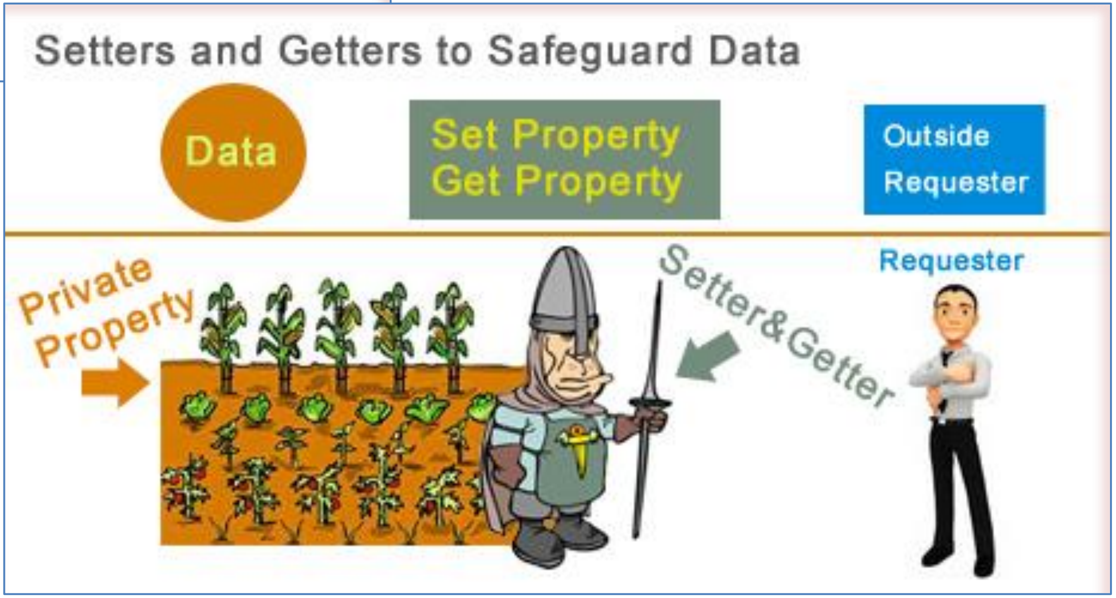
SOLUTION? Put a high fence around my garden, now it is safe! But waite, I can no longer access my own garden.

http://www.evinw.com/w/

Refactoring Spot 7.0: Setters and Getters

SOLUTION: Hire a **private** guard and give him **rules** on who is able to access the garden. Anyone wanting to use the garden must get permission from guard. garden is now **safe** and **accessible**.

Private Property
Setter&Getter



Setters and Getters to Safeguard Data

http://www.evinw.com/w/

Refactoring Spot 7.0: Setters and Getters

SOLUTION: Hire a **private** guard and give him **rules** on who is able to access the garden. Anyone wanting to use the garden must get permission from guard. garden is now **safe** and **accessible**.

Setters and Getters to Safeguard Data

Data
Set Property
Get Property
Outside Requester
Private Property
Setter&Getter
Requester

Encapsulation Step 3: Provide <u>public</u> setter and getter methods to modify and view the fields values.

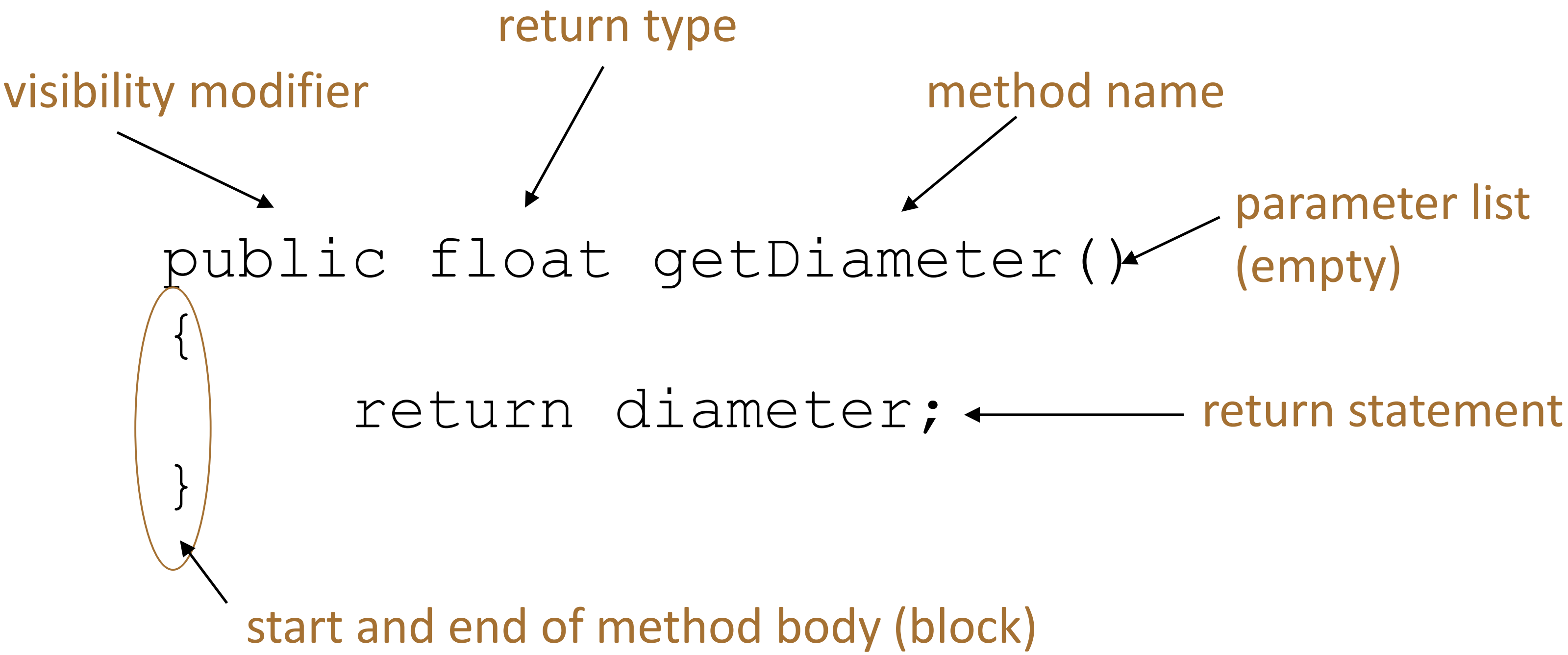
http://www.evinw.com/w/

Getters

- Accessor methods return information about the state of an object.

- A ‘getter’ method is a specific type of accessor method and typically:
  - contains a return statement (as the last executable statement in the method).
  - defines a return type.
  - does NOT change the object state.

Getters



```
public float getDiameter()
{
     return diameter;
}
```

return type

visibility modifier

method name

parameter list (empty)

return statement

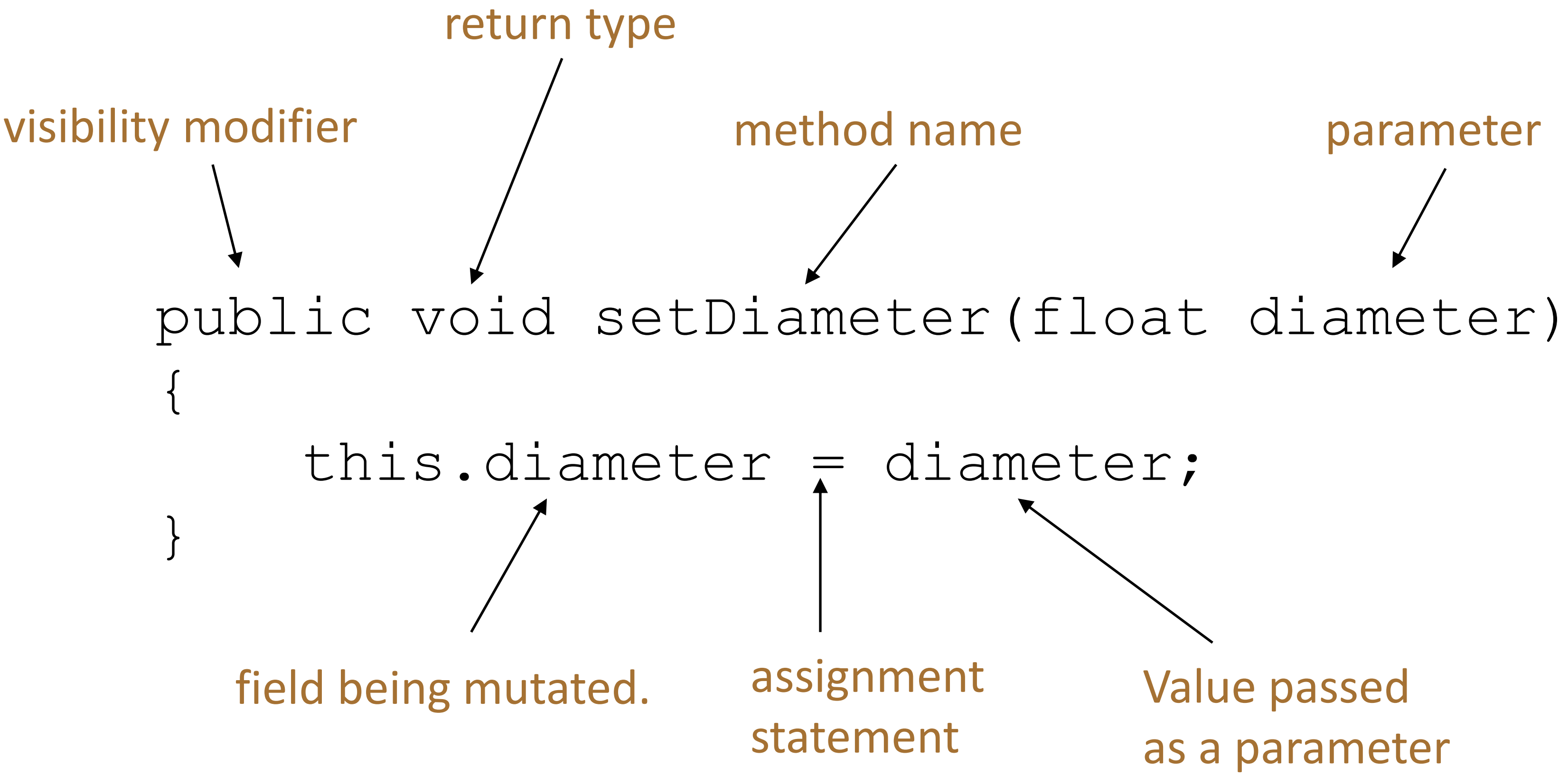
start and end of method body (block)

Setters

- Mutator methods change (i.e. mutate!) an object’s state.

- A ‘setter’ method is a specific type of mutator method and typically:
  - contains an assignment statement
  - takes in a parameter
  - changes the object state.

Setters



return type

visibility modifier

method name

parameter

```
public void setDiameter(float diameter)
{
    this.diameter = diameter;
}
```

field being mutated.

assignment
statement

Value passed
as a parameter

Getters/Setters

- For each instance field in a class, you are normally asked to write:
  - A getter
  - A setter

Refactoring Spot 7.0: Getters

```
public class Spot{
  private float xCoord, yCoord;
  private float diameter;
  private int red, green, blue;

  //constructors…
  //display method…
  // move method…
  // colour methods…

  public float getDiameter(){
    return diameter;
  }
```

```
 public float xCoord(){
     return xCoord;
   }

public float yCoord(){
   return yCoord;
  }

  public int getRed(){
   return red;
  }
```

```
  public int getGreen(){
    return green;
  }

  public int getBlue(){
    return blue;
  }

  public int getGray(){
    return gray;
  }

} //end Spot class
```

Refactoring Spot 7.0: Setters (1 of 2)

```
public class Spot{
  private float xCoord, yCoord;
  private float diameter;
  private int red, green, blue;

  //constructors…
  //display method…
  // move method…
  // colour methods…
  // assessor methods…

 public void setDiameter(float diameter){
    this.diameter = diameter;
 }
```

```
public void setXCoord(float xCoord){
  this.xCoord = xCoord;
}

public void setYCoord(float yCoord){
  this.yCoord = yCoord;
}

public void setRed(int red){
  this.red = red;
}
```

Refactoring Spot 7.0: Setters (2 of 2)

```
  public void setGreen(int green){
    this.green = green;
  }

  public void setBlue(int blue){
    this.blue = blue;
  }

  public void setGray(int gray){
    this.gray = gray;
  }
}
```

Refactoring Spot 7.0

```
Spot sp;

void setup()
{
  size (100,100);
  noStroke();
  sp = new Spot(33, 50, 30);
}

void draw()
{
  background(0);
  sp.colour(255, 0, 0);
  sp.setDiameter(30000);
  sp.display();
}
```

```
class Spot{
  float xCoord, yCoord;
  float diameter;
  int red, green, blue;

  // constructors...
 void display(){
   ellipse(xCoord, yCoord, diameter, diameter);
  }

 void colour(int red, int green, int blue)
  {
   this.red = red;
   this.green = green;
   this.blue = blue;
   fill (red, green, blue);
  }
 // move methods...
}
```

Source: Reas & Fry (2014)

But Our Design Still Smells!

- We have hidden our fields (they are private) and we have getter and setter methods to view/update the fields. We have enforced the Encapsulation rules.

- BUT we can still set the field values to undesirable values!

- We need validation.

Improving the constructor

```
Spot(float xCoord, float yCoord, float diameter)
{
  this.xCoord = xCoord;
  this.yCoord = yCoord;
  this.diameter = diameter;
}
```

```
Spot(float xCoord, float yCoord, float diameter)
{
  this.xCoord = xCoord;
  this.yCoord = yCoord;
  if ((diameter > 0) && (diameter < 500)){
     this.diameter = diameter;
  }
  else{
     this.diameter = 10;
  }
}
```

Note: in the constructor, you typically set the field to a default value if invalid data was entered.

Improving the setter / mutator

```
public void setDiameter(float diameter){
    if ((diameter > 0) && (diameter < 500)){
        this.diameter = diameter;
    }
}
```

Note: The validation done at constructor level must be repeated at setter level for that field → data integrity!
However, in setter methods, you typically do not update the field’s value if invalid data was entered (notice how the “else” part of the “if” is not there).

Refactoring Spot 7.0: Validation

- When the fields, red, green, blue and gray are being updated, you could validate the data to ensure they are being set to a number between 0 and 255 (inclusive).

- You could also validate the xCoord and yCoord to values that you wouldn’t expect them to exceed (in both directions).

Questions?

References

- Reas, C. & Fry, B. (2014) Processing – A Programming Handbook for Visual Designers and Artists, 2nd Edition, MIT Press, London.

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Department of Computing and Mathematics
http://www.wit.ie/