Public and private access

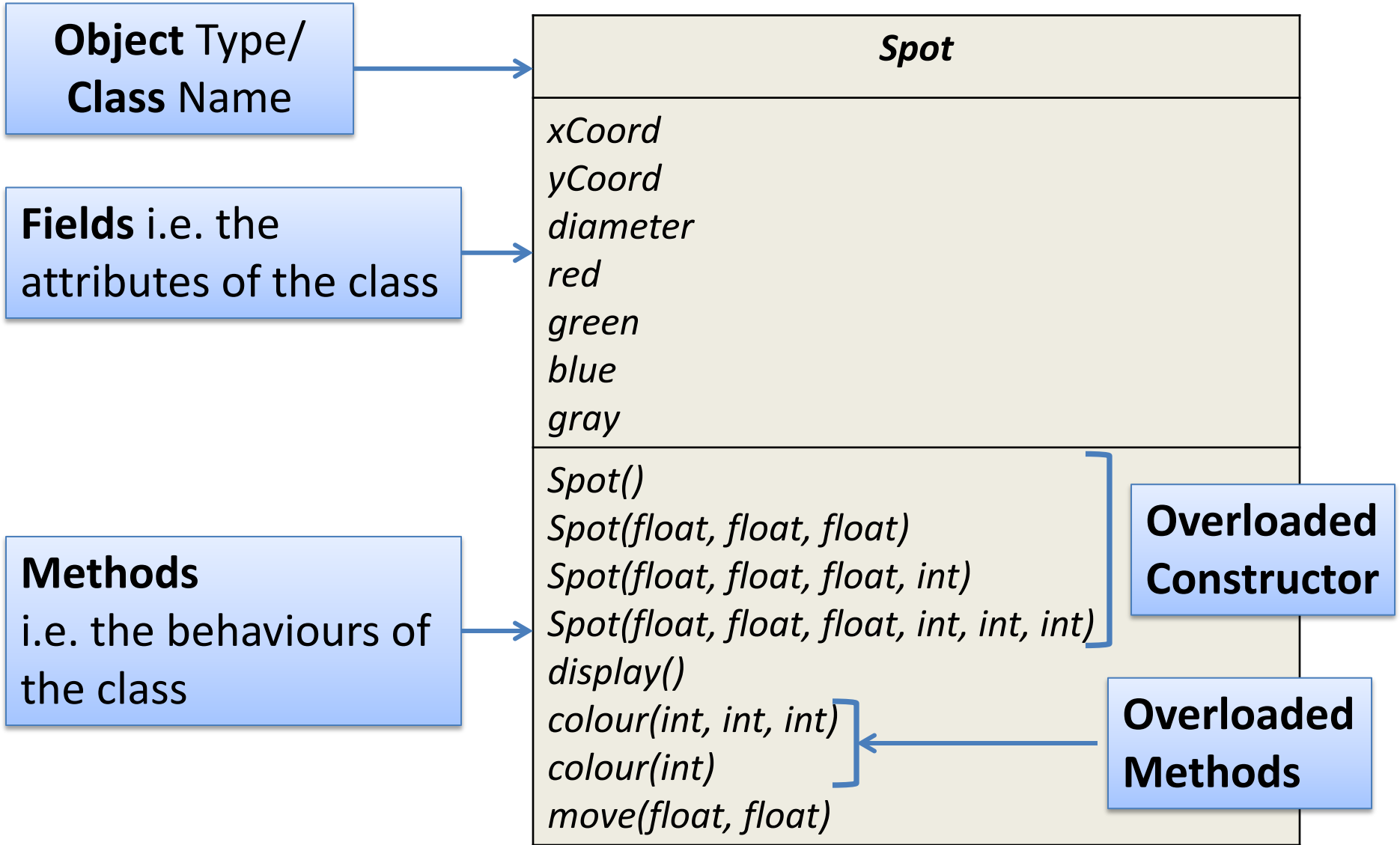
Why do we need private fields and public methods?

Produced by: Ms. Mairead Meagher
Dr. Siobhán Drohan

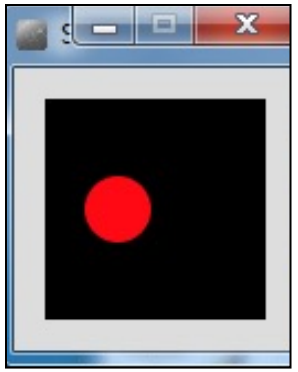
Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

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

Class Diagram for Spot Version 2



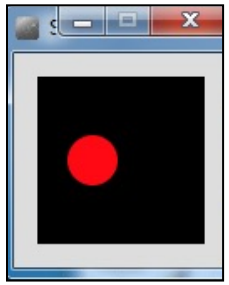
Spot Class – Version 2



```
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 2



```
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 2**

```
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 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 i.e. 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 Object Oriented 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 – **steps 1-3**

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 <u>private</u>.**
3. **Access** the fields only through the methods of their current class.	**Provide <u>public</u> set**ter **and get**ter 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: **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).

We will focus on public and private.

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

Refactoring Spot : **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.

We just changed the class access level to **public**
(default is **package**).

Filename: Spot

Refactoring Spot : 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…
}
```

However, as the default access level is **package** → our methods and fields are all **package** level access.

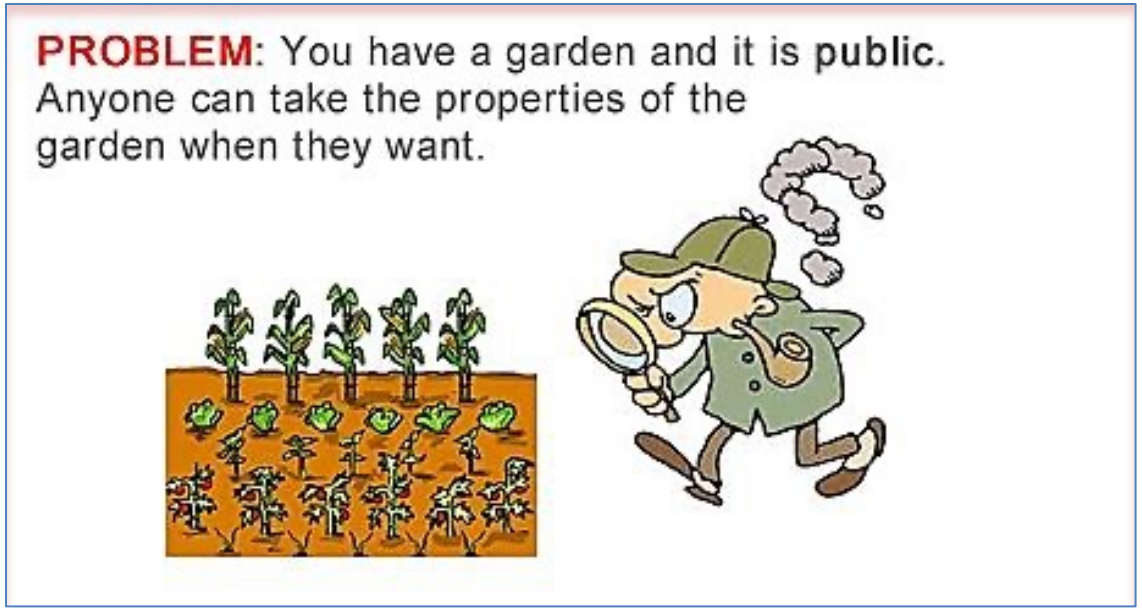
Problem: this breaks **Encapsulation step 2** i.e. the fields of a class should be **private**.

Refactoring Spot: 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…
}
```

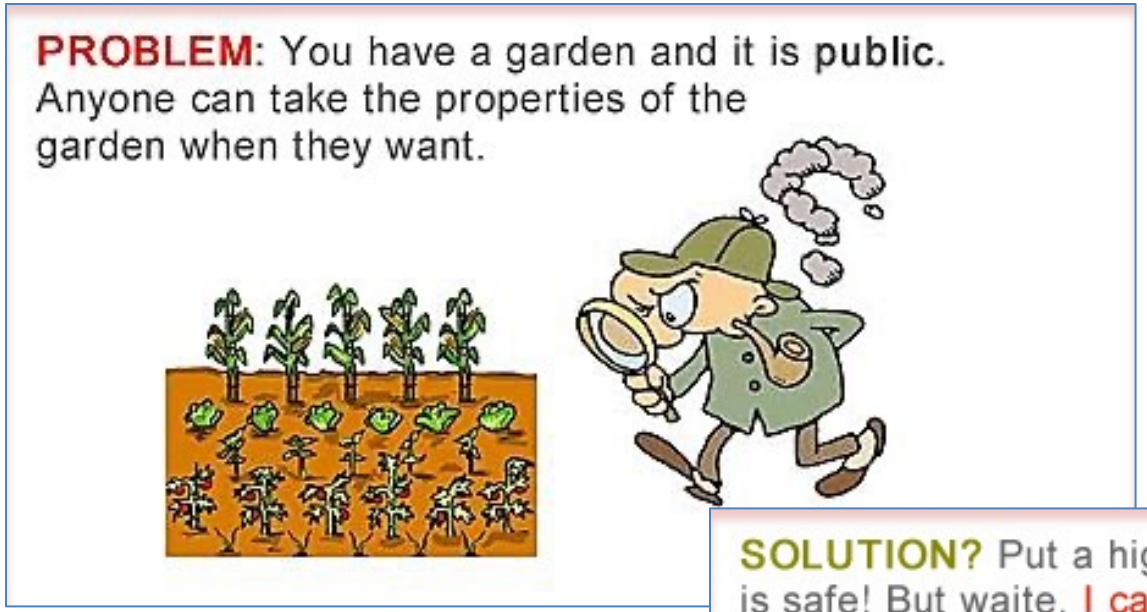
To fix **Encapsulation step 2**,
we **declare all the fields** with **private** access.

Access Modifiers



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

Access Modifiers



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…
}
```

The **private** fields are not viewable or updatable outside the class **Spot**. Other classes don't know these exist.

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

Refactoring Spot : **Setters and Getters**

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

Refactoring Spot 2: **Setters and Getters**

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

Refactoring Spot 2: **Setters and Getters**

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

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

Refactoring Spot 2: **Get**ters

```
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 getXCoord(){
    return xCoord;
  }

public float getYCoord(){
   return yCoord;
  }

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

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

  public int getBlue(){
    return blue;
  }

} //end Spot class
```

Refactoring Spot 2: **Set**ters

```
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;
}
```

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

public void setBlue (int blue){
```

Spot Class – Version 2

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

Before refactoring,
we directly accessed the diameter field…
this broke Encapsulation rules.

Source: Reas & Fry (2014)

Refactoring Spot 2 – getters and setters

```
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();
}
```

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

  // constructors…
  // display method…
  // colour methods…
  // move methods…
  //getter methods…
  //setter methods…
  public void setDiameter (float diameter) {
     this.diameter = diameter;
  }
}
```

Now we update via the appropriate setter

Questions?

References

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