Programming Fundamentals 1

Produced by

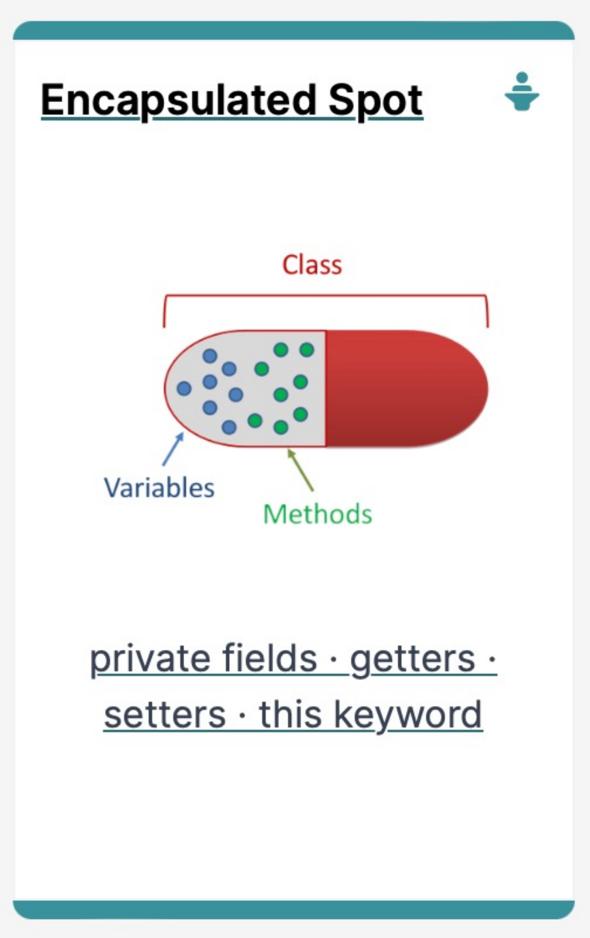
Mr. Dave Drohan
(david.drohan@setu.ie)
Dr. Siobhán Drohan
Ms. Mairead Meagher

Department of Computing & Mathematics
South East Technological University
Waterford, Ireland

setu.ie

IntelliJ and Spot

Encapsulation and Spot



Encapsulated Spot

private fields · getters · setters · this keyword

Agenda

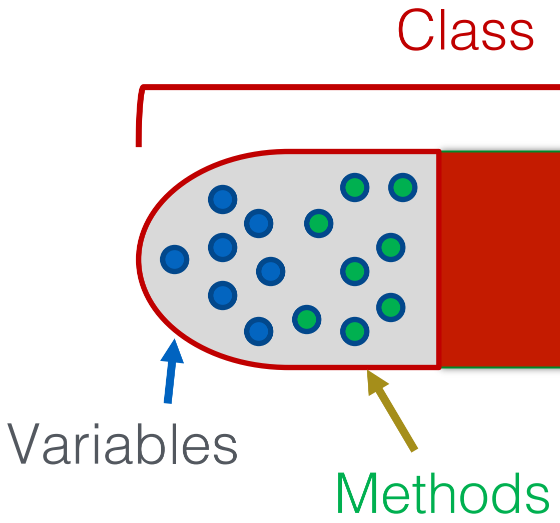
□ What is Encapsulation?

□ Spot and Encapsulation

□ Basic Spot Class

□ The `this` keyword

What is Encapsulation?


Class
Variables
Methods

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

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

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 private**
3. Access the fields only through the methods of their current class	**Provide** public **getter and setter** methods to modify and view the fields values

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)

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 these this semester

Spot and Encapsulation

Spot and Encapsulation

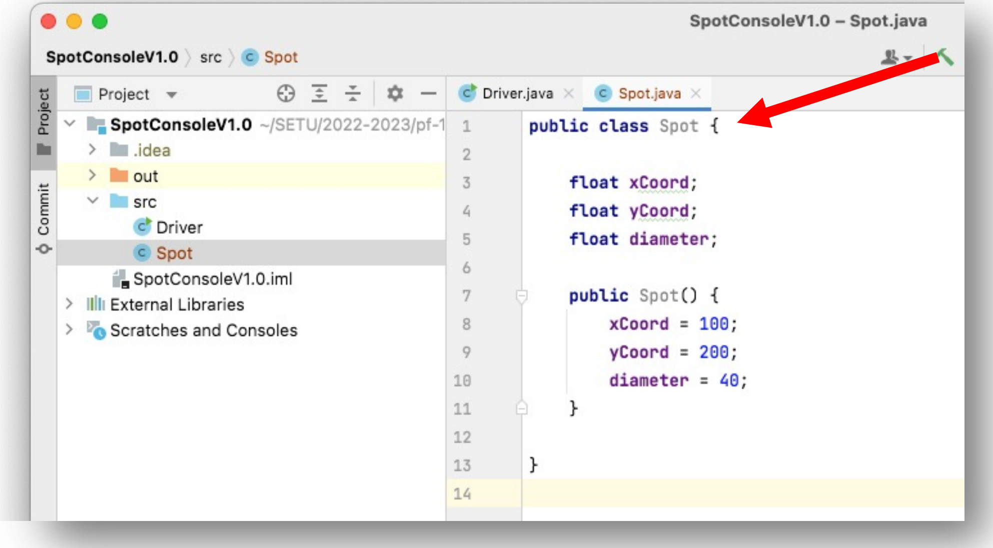
Step 1

Wrap the data (fields) and code acting on the data (methods) together as single unit

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`
	`}`

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



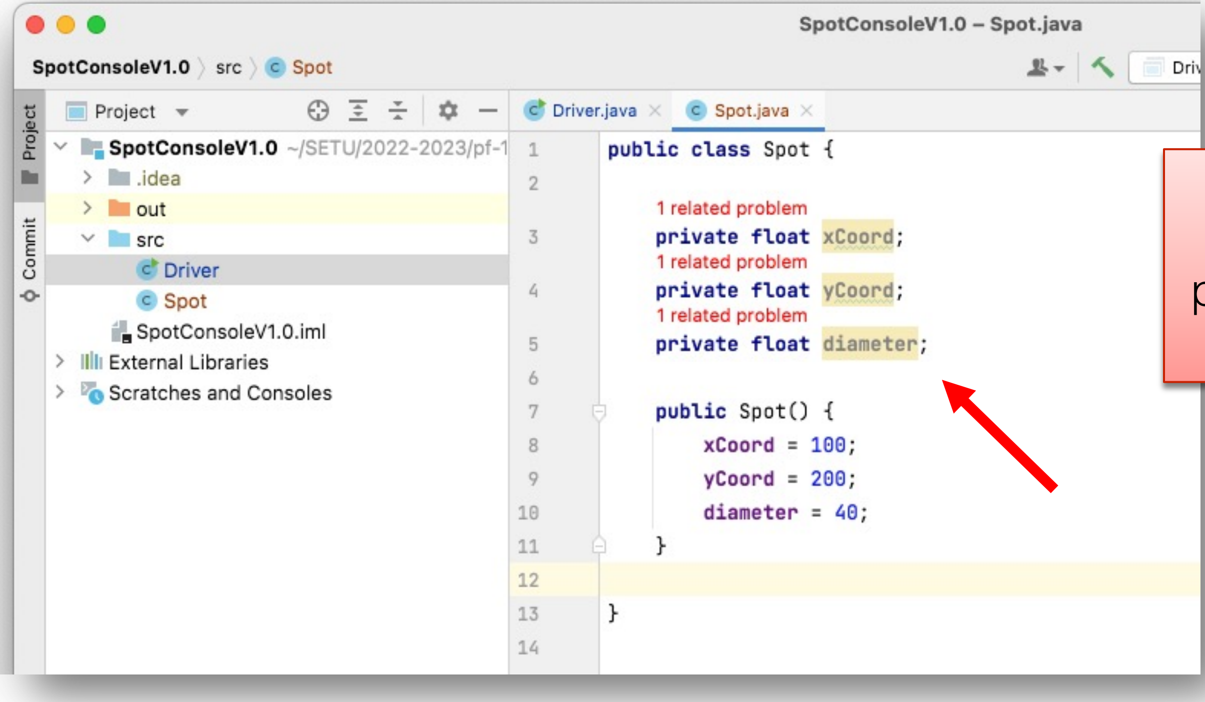
```java
public class Spot {

    float xCoord;
    float yCoord;
    float diameter;

    public Spot() {
        xCoord = 100;
        yCoord = 200;
        diameter = 40;
    }

}
```

Spot and Encapsulation

Step 2

Hide the data (fields) from other Classes

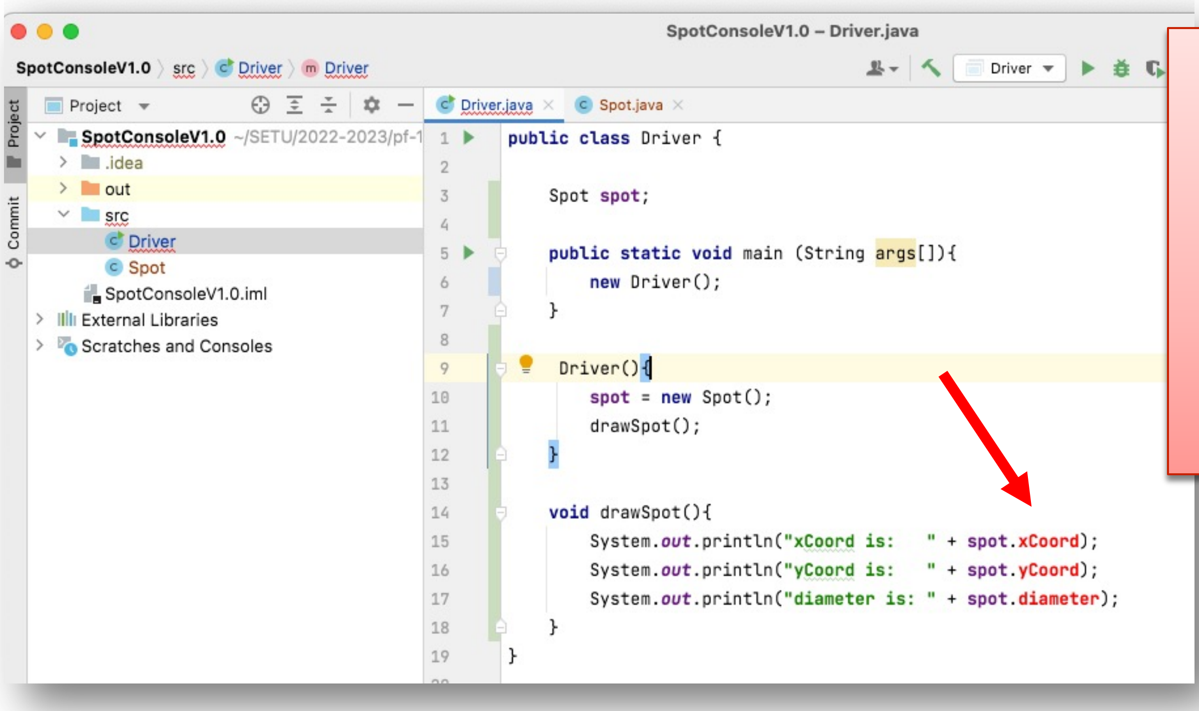
Encapsulation Step	Approach in Java
2. Hide the fields from other classes	Declare the fields of a class as private



```
public class Spot {

    1 related problem
    private float xCoord;
    1 related problem
    private float yCoord;
    1 related problem
    private float diameter;

    public Spot() {
        xCoord = 100;
        yCoord = 200;
        diameter = 40;
    }

}
```

Encapsulation step 2
We have made our fields private, however our app is no longer compiling!

Encapsulation Step	Approach in Java
2. Hide the fields from other classes	Declare the fields of a class as private



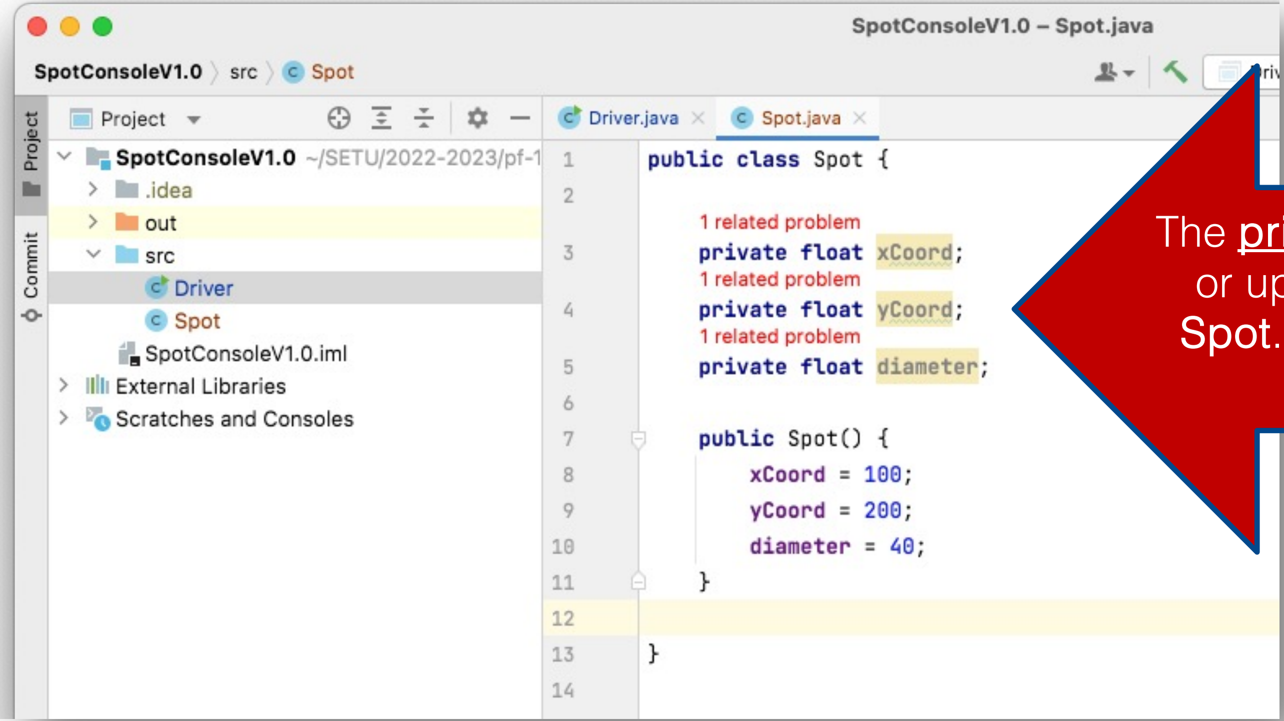
```java
public class Driver {

    Spot spot;

    public static void main (String args[]){
        new Driver();
    }

    Driver(){
        spot = new Spot();
        drawSpot();
    }

    void drawSpot(){
        System.out.println("xCoord is:   " + spot.xCoord);
        System.out.println("yCoord is:   " + spot.yCoord);
        System.out.println("diameter is: " + spot.diameter);
    }
}
```

Encapsulation step 2

The problem lies in the Driver class:

- We are trying to directly access fields that are now private.
- These fields are no longer visible in Driver.

Encapsulation Step	Approach in Java
2. Hide the fields from other classes	Declare the fields of a class as private



```java
public class Spot {

    private float xCoord;
    private float yCoord;
    private float diameter;

    public Spot() {
        xCoord = 100;
        yCoord = 200;
        diameter = 40;
    }

}
```

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

Spot and Encapsulation

Step 3

Access the data (fields) *only* through the methods of their current class

Solution: Getters and Setters

Encapsulation Step 3

Provide public getter and setter methods to modify and view the fields values.

Getters (Accessor Methods)

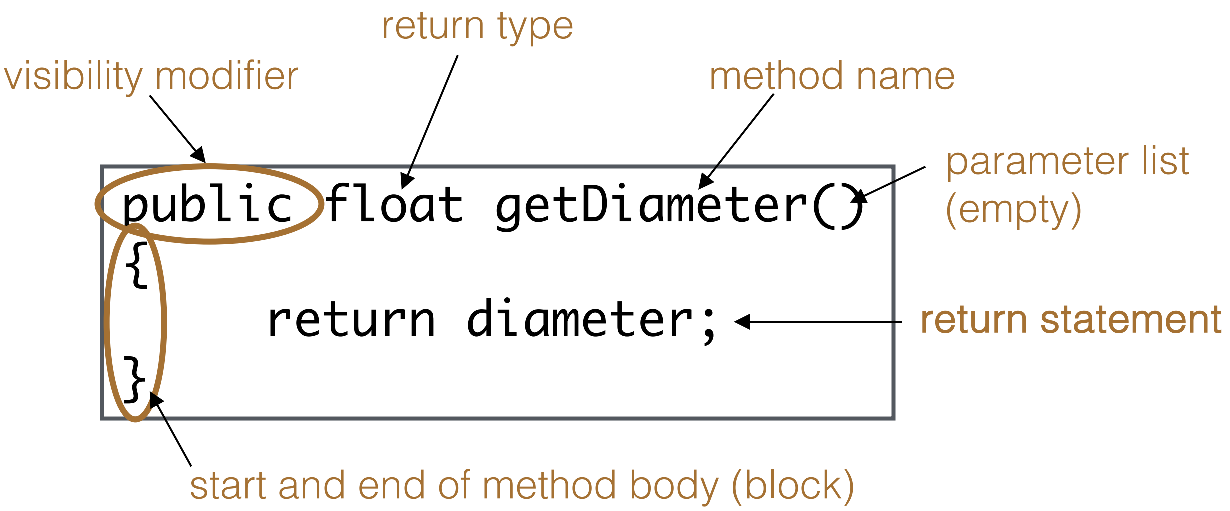
❑Accessor methods

- return information about the **state** of an object i.e. the values stored in the fields

❑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

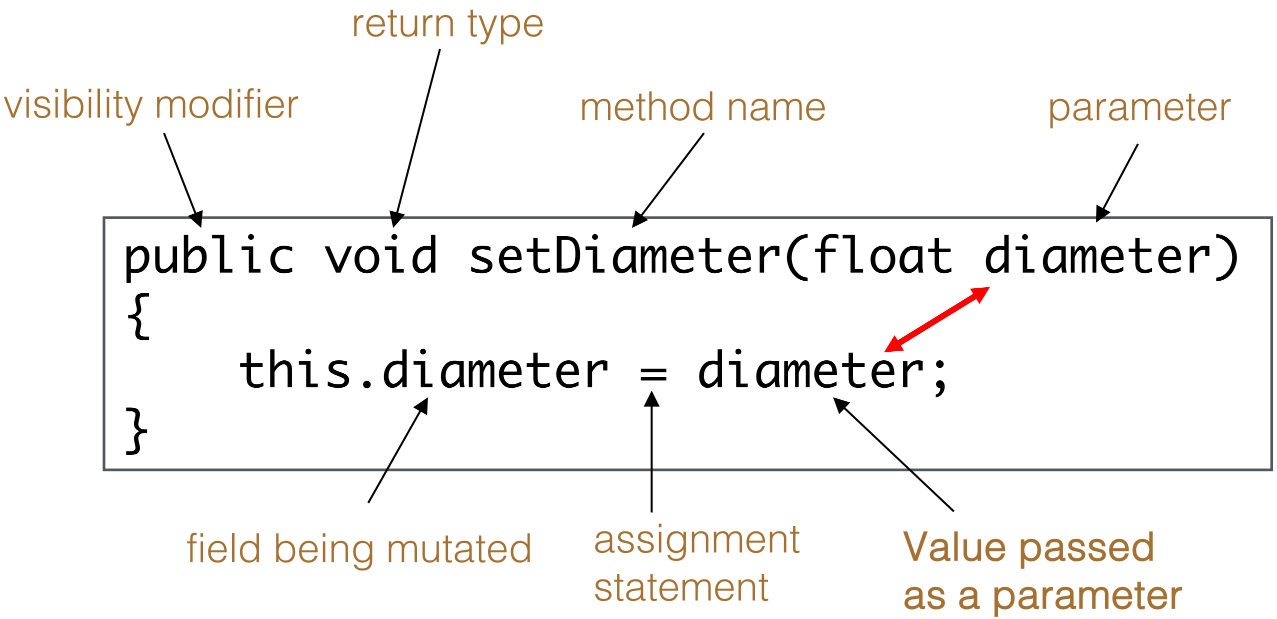
start and end of method body (block)

Setters (Mutator methods)

- ❑ 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
    - *Return* statement
  - A setter
    - *Assignment* statement

Encapsulation Step	Approach in Java
3. Access the fields only through the methods of their current class.	**Provide** public **get**ter **and set**ter methods to modify and view the fields values.

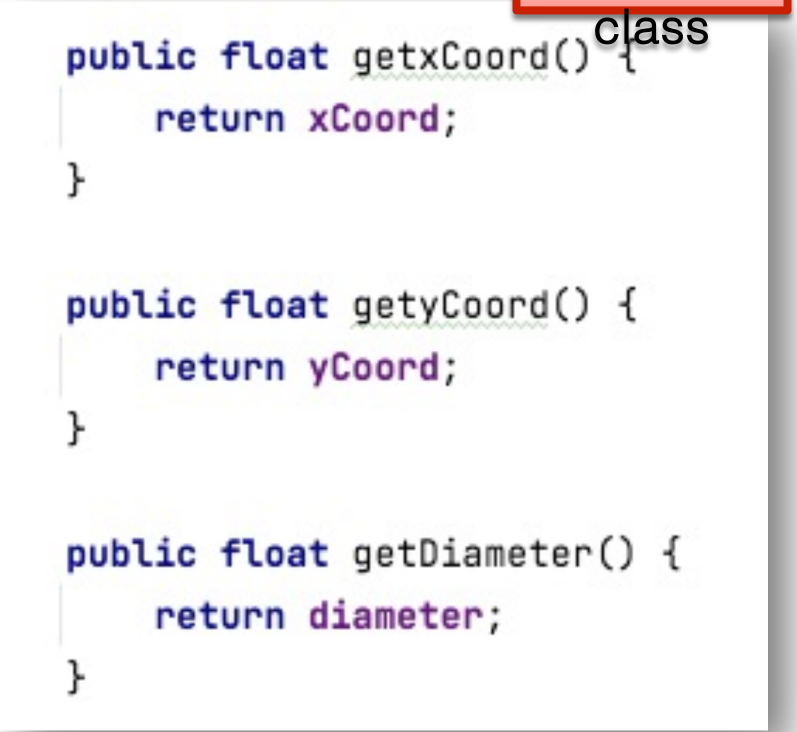
Spot and Encapsulation

Step 3

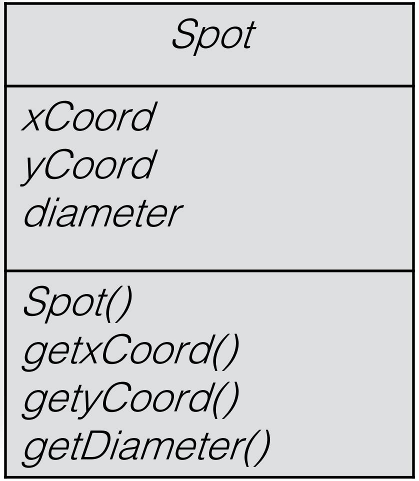
Getters

Encapsulation Step 3: Provide public getter methods to view the fields values.

Spot class



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

public float getyCoord() {
    return yCoord;
}

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



Spot

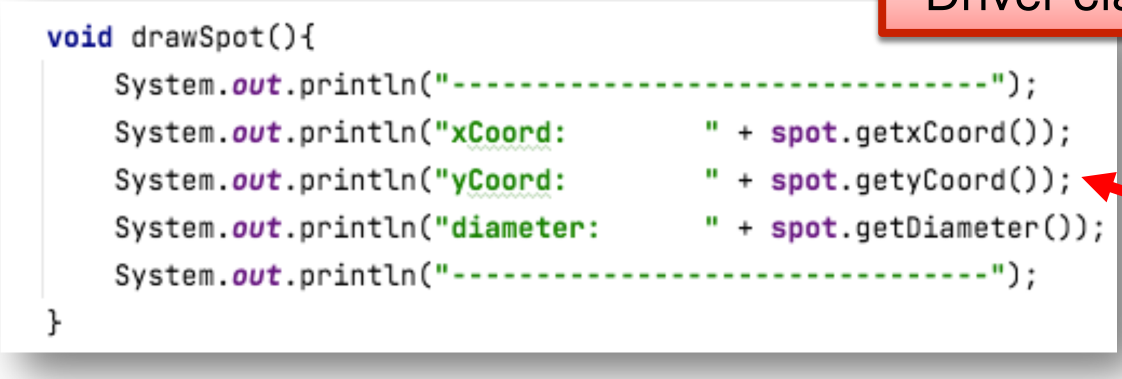
xCoord
yCoord
diameter

Spot()
getxCoord()
getyCoord()
getDiameter()

Encapsulation Step 3:

Use these new getter methods to view the fields values.

Driver class



```
void drawSpot(){
    System.out.println("--------------------------------");
    System.out.println("xCoord:        " + spot.getxCoord());
    System.out.println("yCoord:        " + spot.getyCoord());
    System.out.println("diameter:      " + spot.getDiameter());
    System.out.println("--------------------------------");
}
```

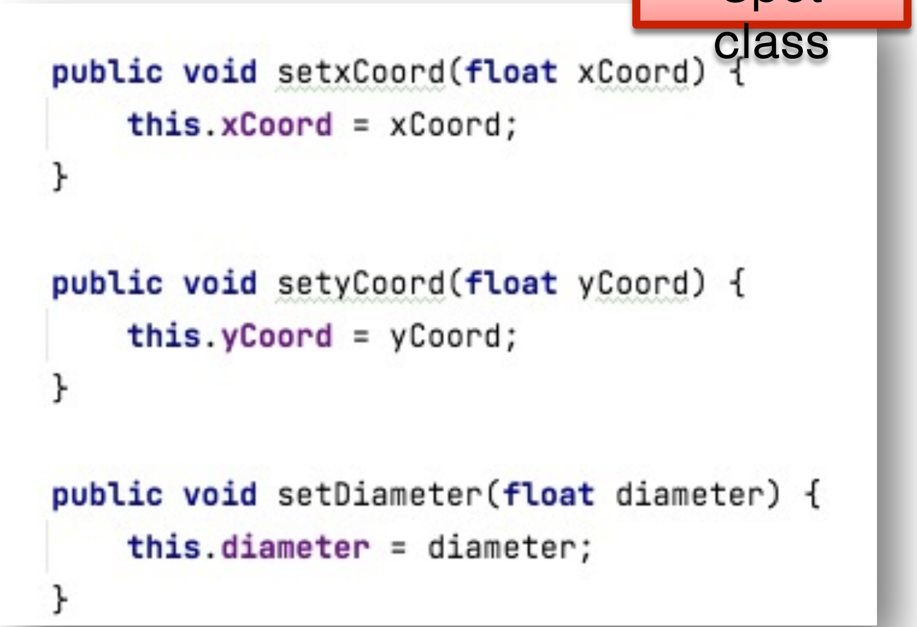
Spot and Encapsulation

Step 3

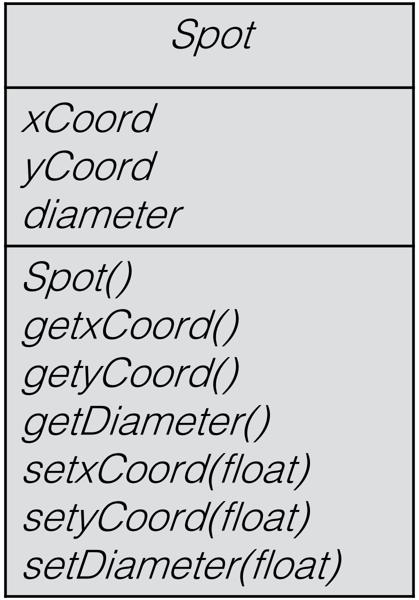
Setters

Encapsulation Step 3: Provide public setter methods to update the fields values.

Spot class



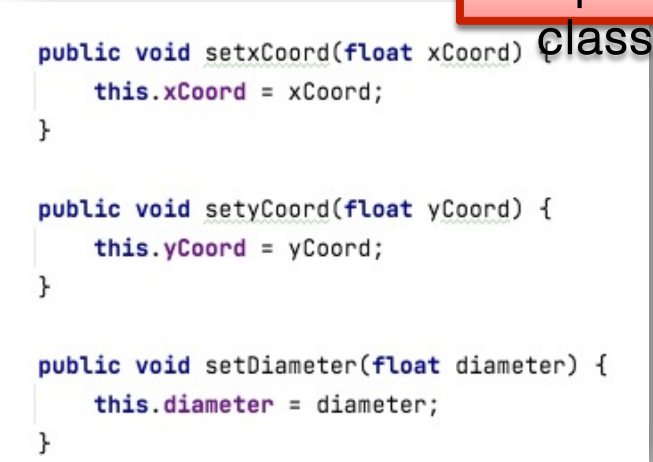
```
public void setxCoord(float xCoord) {
    this.xCoord = xCoord;
}

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

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



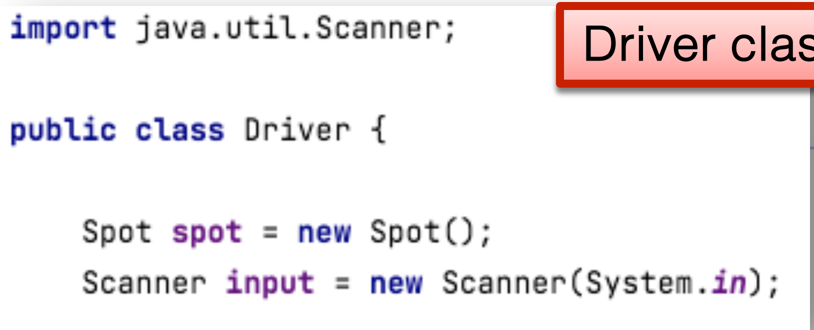
New values for xCoord, yCoord, diameter…

- ❑ To demonstrate the use of these mutator/setter methods, we need to update the **Spot** variables with new values.
- ❑ The easiest way to get new values is to ask the user to enter them on the console.
- ❑ To do this, we will use the `Scanner` class (which we will cover in more detail next week).

Spot class



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

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

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

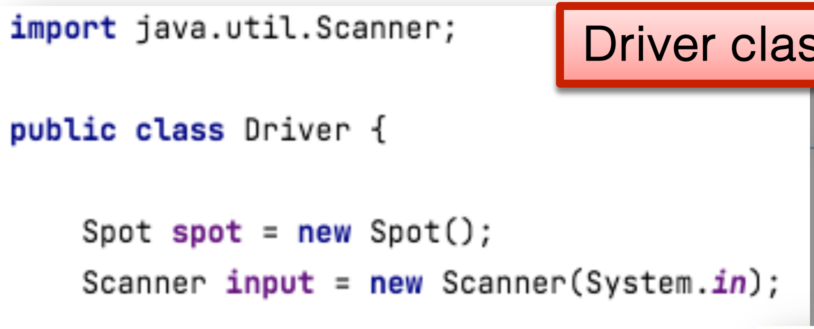


```java
import java.util.Scanner;

public class Driver {

    Spot spot = new Spot();
    Scanner input = new Scanner(System.in);
```

Driver class

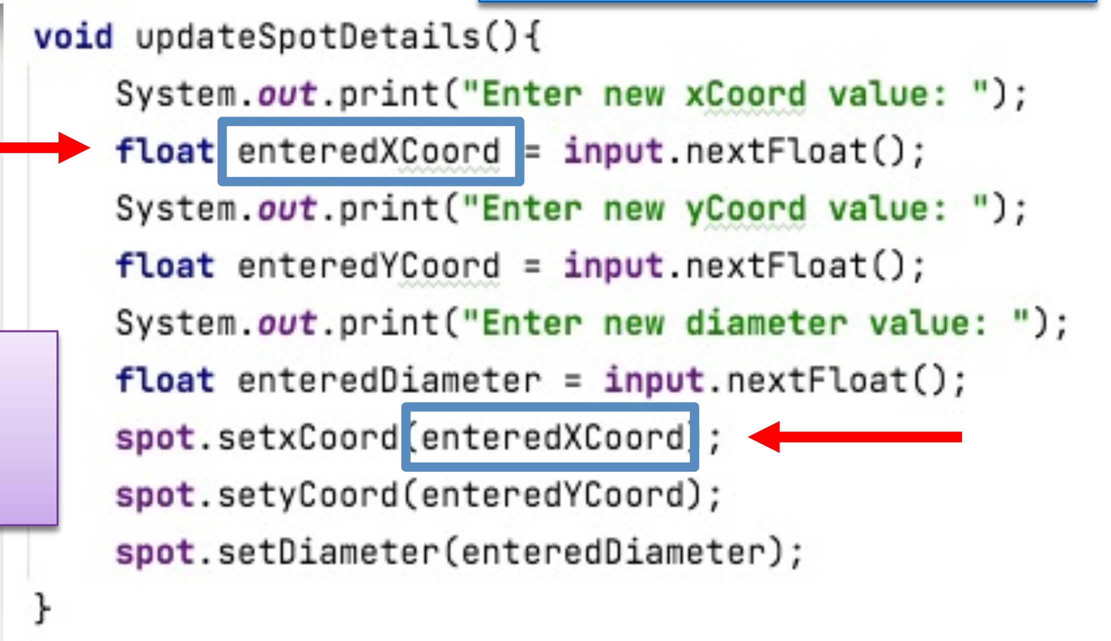
Scanner Class to read from the console

Driver class



```
import java.util.Scanner;

public class Driver {

    Spot spot = new Spot();
    Scanner input = new Scanner(System.in);
```

Method to ask the user to enter new values for the three fields.

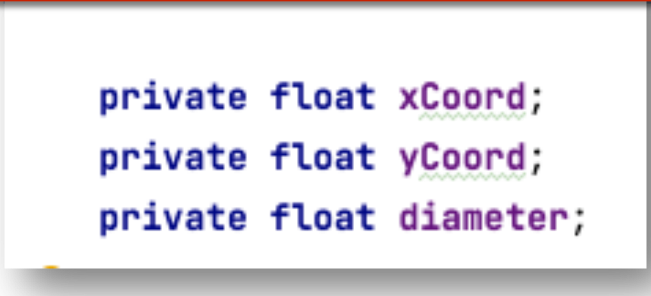


```
void updateSpotDetails(){
    System.out.print("Enter new xCoord value: ");
    float enteredXCoord = input.nextFloat();
    System.out.print("Enter new yCoord value: ");
    float enteredYCoord = input.nextFloat();
    System.out.print("Enter new diameter value: ");
    float enteredDiameter = input.nextFloat();
    spot.setxCoord(enteredXCoord);
    spot.setyCoord(enteredYCoord);
    spot.setDiameter(enteredDiameter);
}
```

The setters are then called to update the values in the spot object.

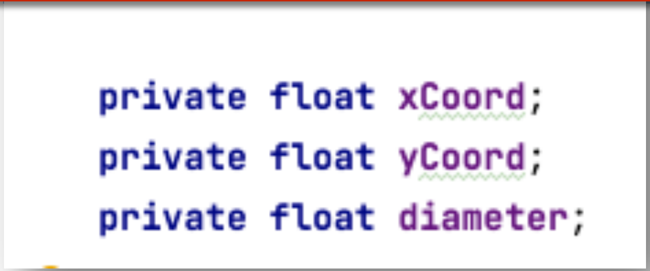
The `this` Keyword

In **Spot**, there are three private instance fields:



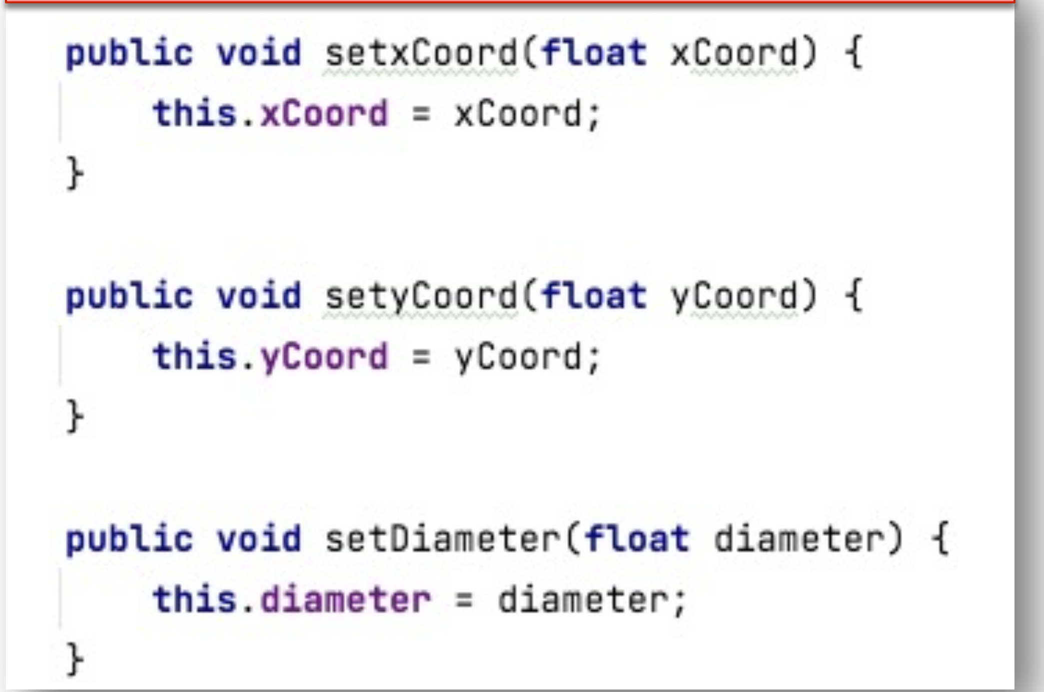
```
private float xCoord;
private float yCoord;
private float diameter;
```

In Spot, there are three private instance fields:



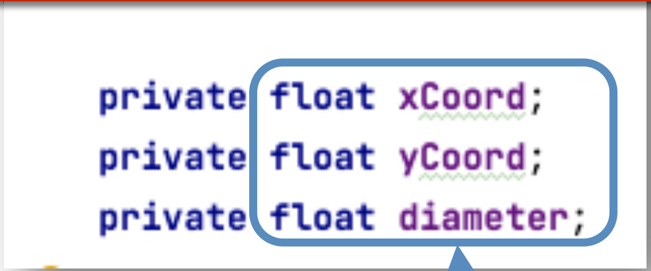
```
private float xCoord;
private float yCoord;
private float diameter;
```

In **Spot**, there is a setter for each of these fields:



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

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

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

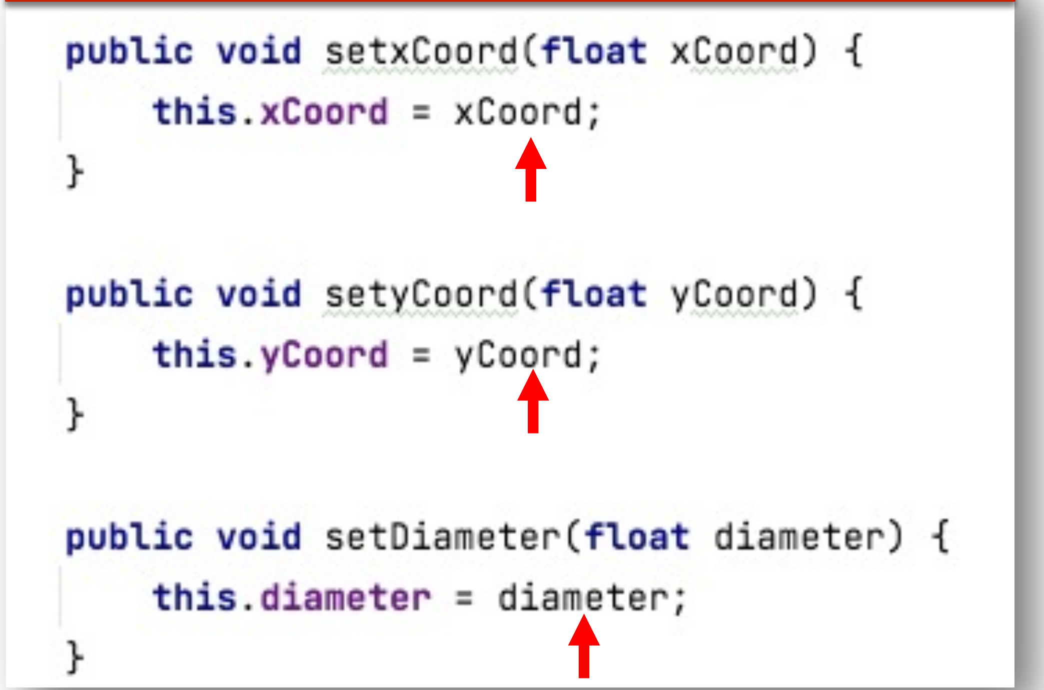
In Spot, there are three private instance fields:



```
private float xCoord;
private float yCoord;
private float diameter;
```

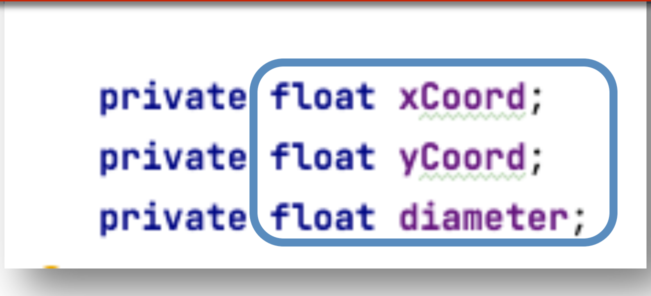
The instance fields (global) are named the same as the parameters for the setters (which are local fields).

In **Spot**, there is a setter for each of these fields:



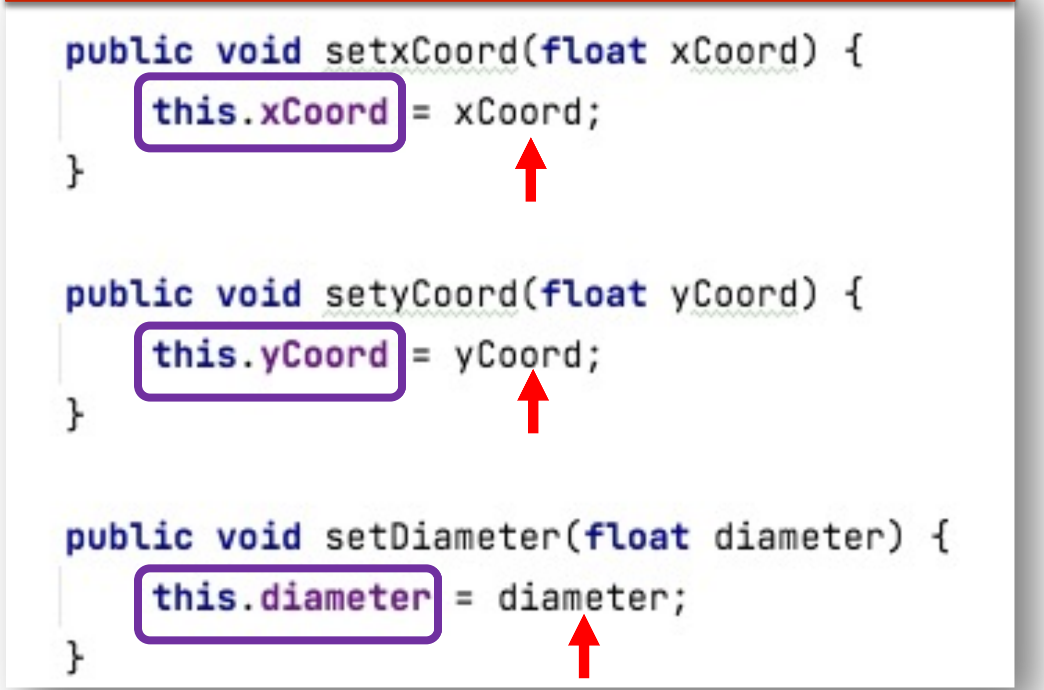
```
public void setxCoord(float xCoord) {
    this.xCoord = xCoord;
}

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

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

In Spot, there are three private instance fields:



```
private float xCoord;
private float yCoord;
private float diameter;
```

In **Spot**, there is a setter for each of these fields:

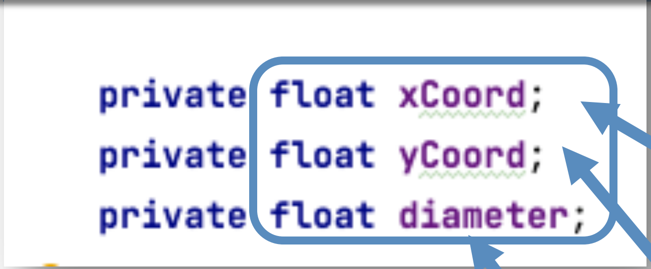


```
public void setxCoord(float xCoord) {
    this.xCoord = xCoord;
}

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

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

This is called name overloading.

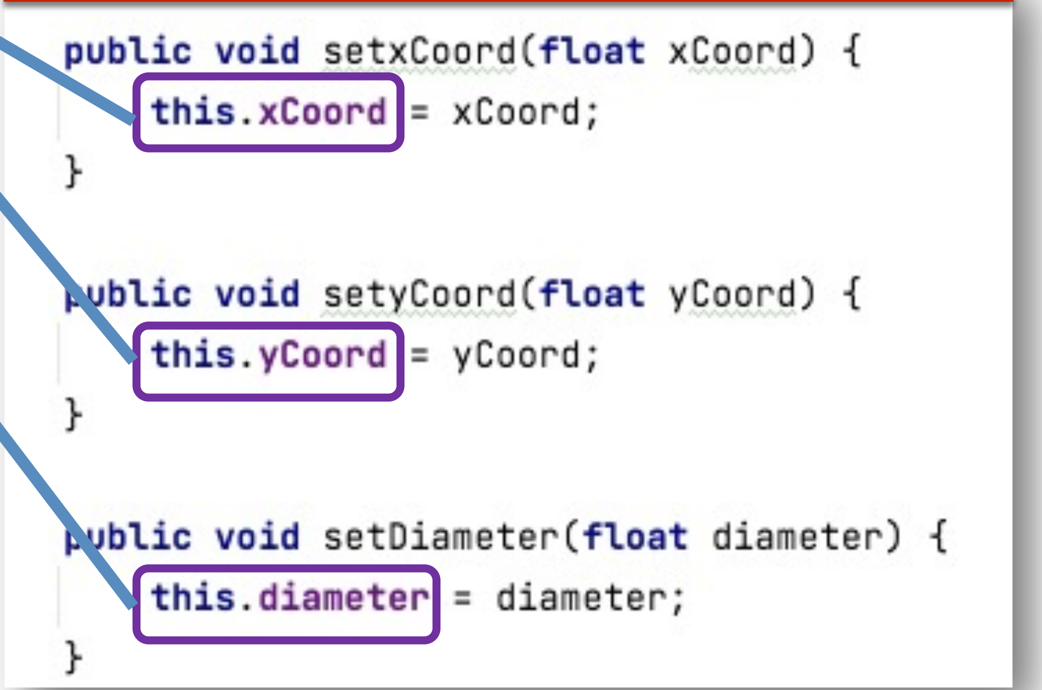
We use `this.` to distinguish between local and global variables.

In Spot, there are three private instance fields:



```
private float xCoord;
private float yCoord;
private float diameter;
```

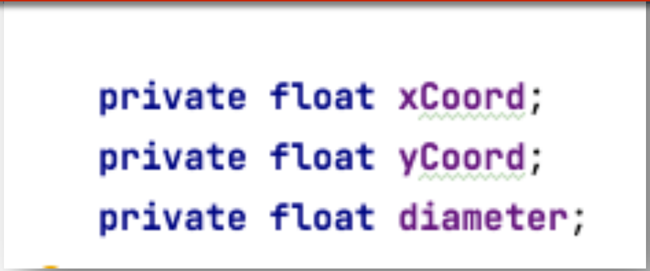
In **Spot**, there is a setter for each of these fields:



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

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

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

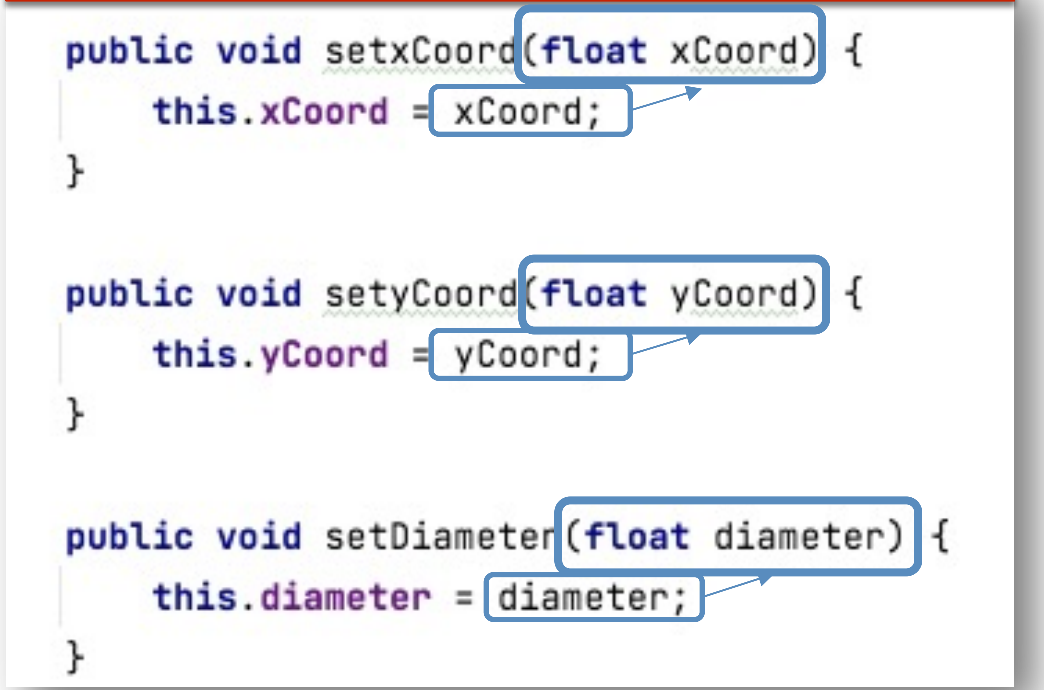
`this.` refers to the **current** objects fields i.e. the global ones.

In Spot, there are three private instance fields:



```
private float xCoord;
private float yCoord;
private float diameter;
```

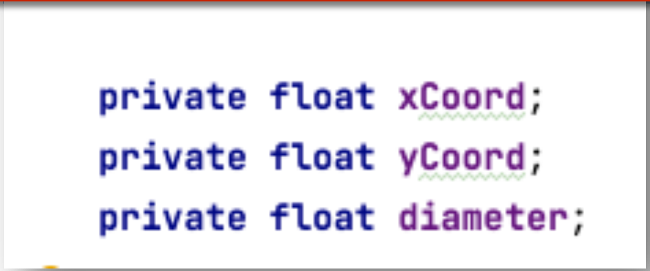
In **Spot**, there is a setter for each of these fields:



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

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

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

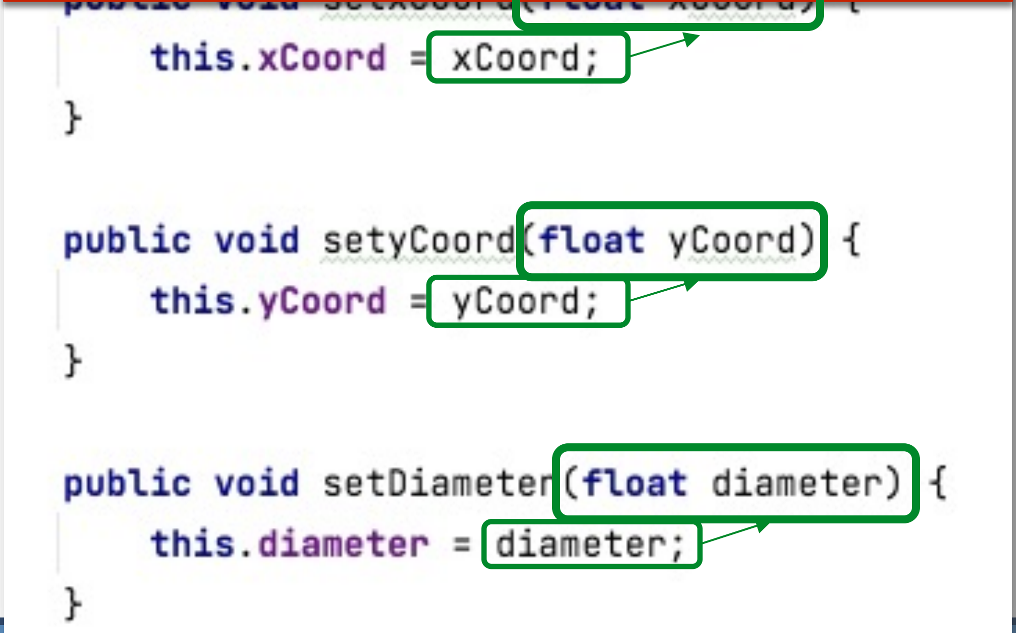
The variables without the `this.` are the **local** ones that are destroyed when the method is finished running i.e. the local variables.

In Spot, there are three private instance fields:



```
private float xCoord;
private float yCoord;
private float diameter;
```

In Spot, there is a setter for each of these fields.



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

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

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

The variables without the this. are the local ones that are destroyed when the method is finished running i.e. the local ones.

Questions?

Thanks.