Recap of OO concepts

Objects, classes, methods and more.

Produced by: Mairead Meagher
Dr. Siobhán Drohan

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

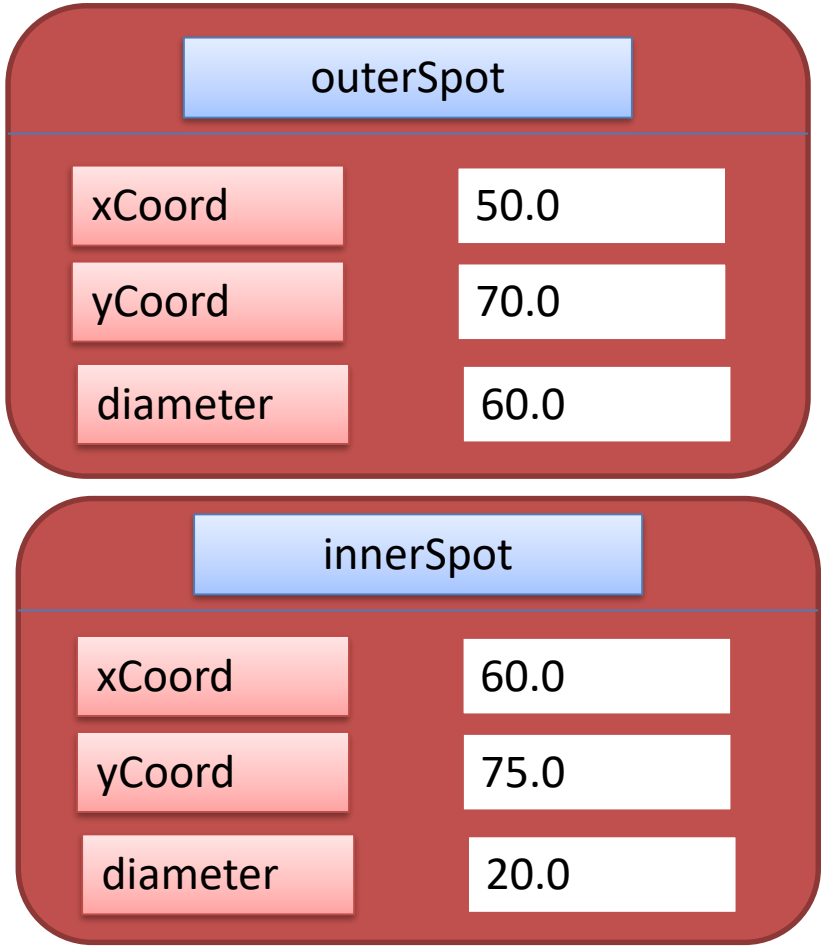
Department of Computing and Mathematics
http://www.wit.ie/

Classes and Objects

- A class defines a group of related methods (functions) and fields (variables).
- An object is a single instance of a class i.e. an object is created from a class.
- Objects can be related to real-world artefacts.
- Many objects can be constructed from a single class definition.
- Each object must have a unique name within the program.

Spot Class

Spot
xCoord *yCoord* *diameter*
Spot() *Spot(float, float, float)* *display()* *getXCoord()* *getYCoord()* *getDiameter()* *setXCoord(float)* *setYCoord(float)* *setDiameter(float)* *toString()*



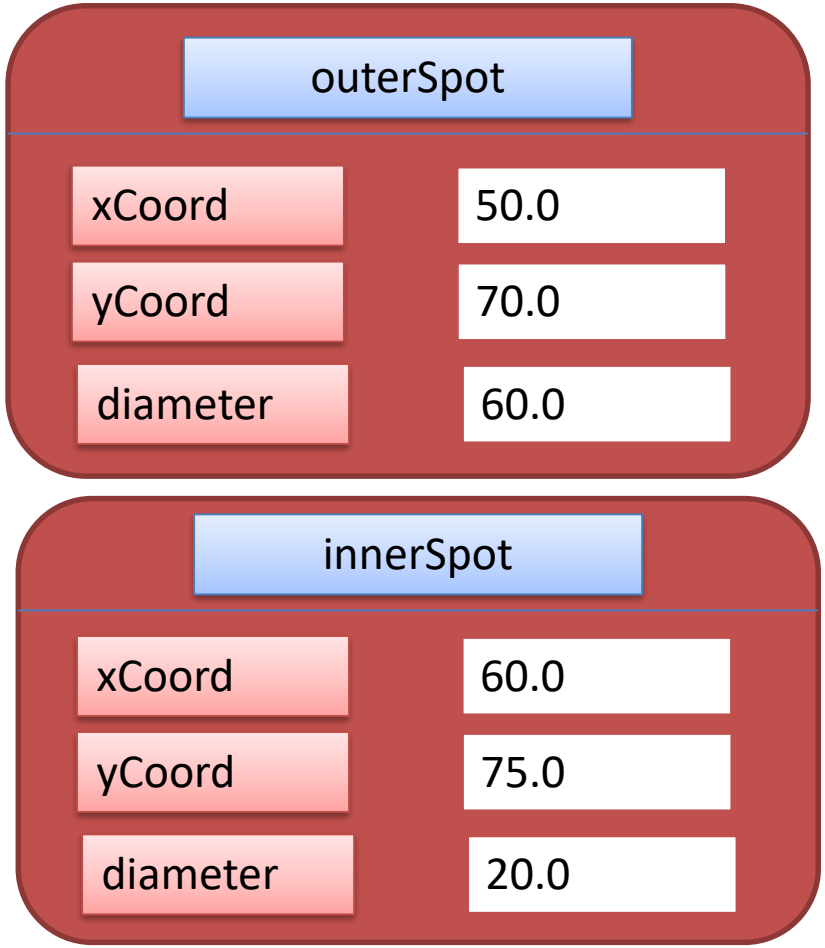
Two objects. Each has a unique name and it's own copy (values) of the fields.

Object State

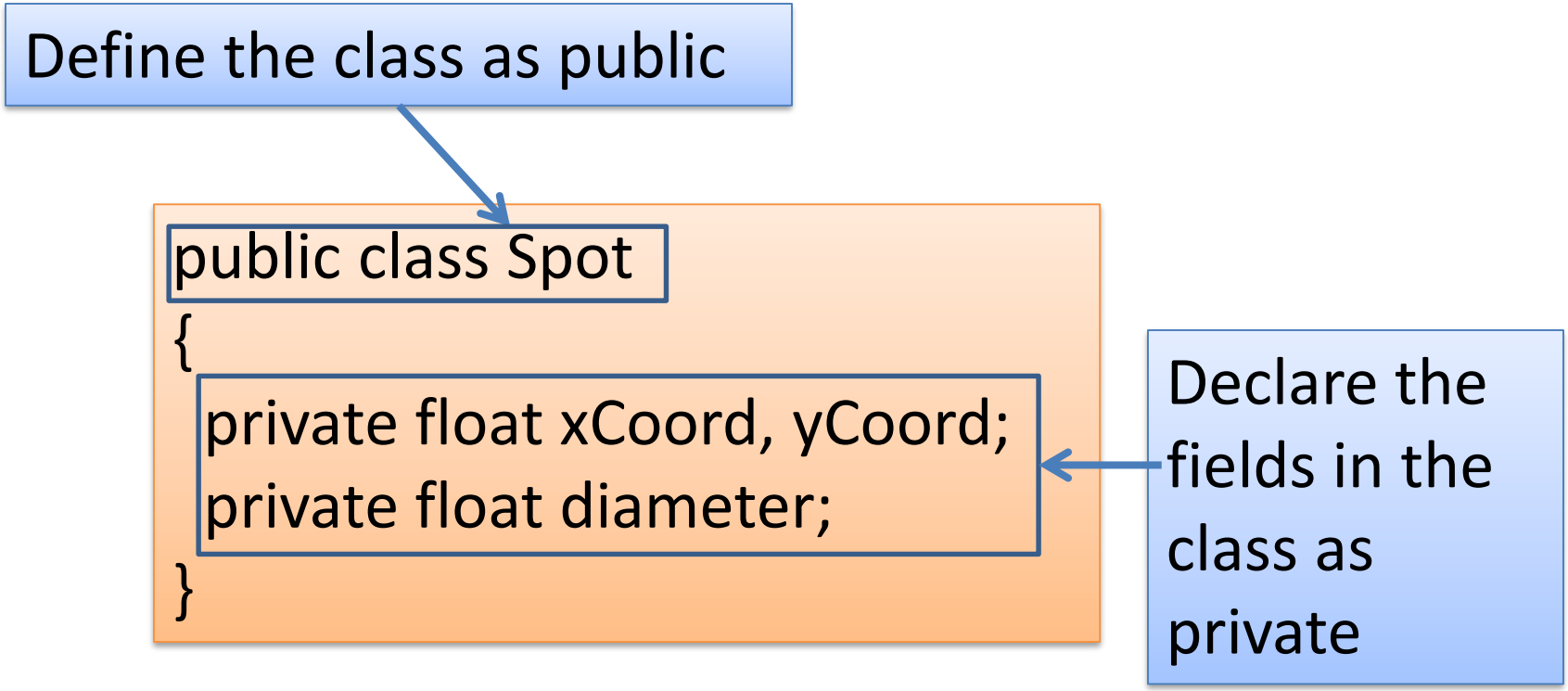
There are two objects of type Spot.

Each object has a unique name:
- innerSpot
- outerSpot

Each object has a different **object state**:
- each object has it's own copy of the fields (xCoord, yCoord, diameter) in memory and has it's own data stored in these fields.



Spot Class



Source: Reas & Fry (2014)

Constructor(s)

- **Spot()** is the default constructor that is called to build an object in memory.
- A constructor is a method that has the same name as the class but has no return type.

```
public class Spot
{
  private float xCoord, yCoord;
  private float diameter;

  //Default Constructor
  public Spot()
  {
  }

  //Constructor to initialise all instance fields.
  public Spot(float xCoord, float yCoord, float diameter)
  {
    this.xCoord = xCoord;
    this.yCoord = yCoord;
    this.diameter = diameter;
  }
```

Getters

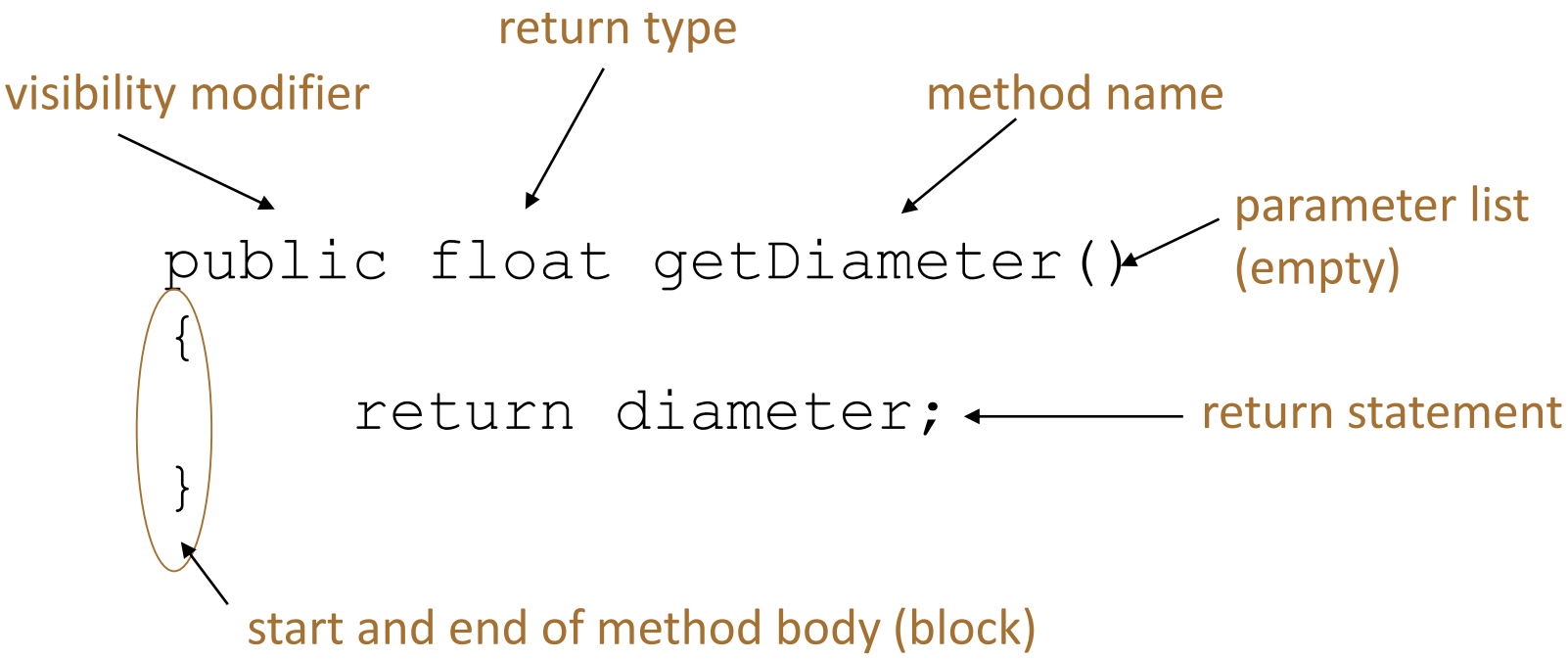
Accessor methods return information about the state of an object.

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.

```
public class Spot
{
  private float xCoord, yCoord;
  private float diameter;

  //constructors…

//     getters      //
 public float getDiameter(){
    return diameter;
 }

 public float getXCoord(){
   return xCoord;
 }

 public float getYCoord(){
   return yCoord;
 }
```

Getters


return type
visibility modifier
method name
parameter list
(empty)
public float getDiameter()
{
return diameter;
return statement
}
start and end of method body (block)

Setters

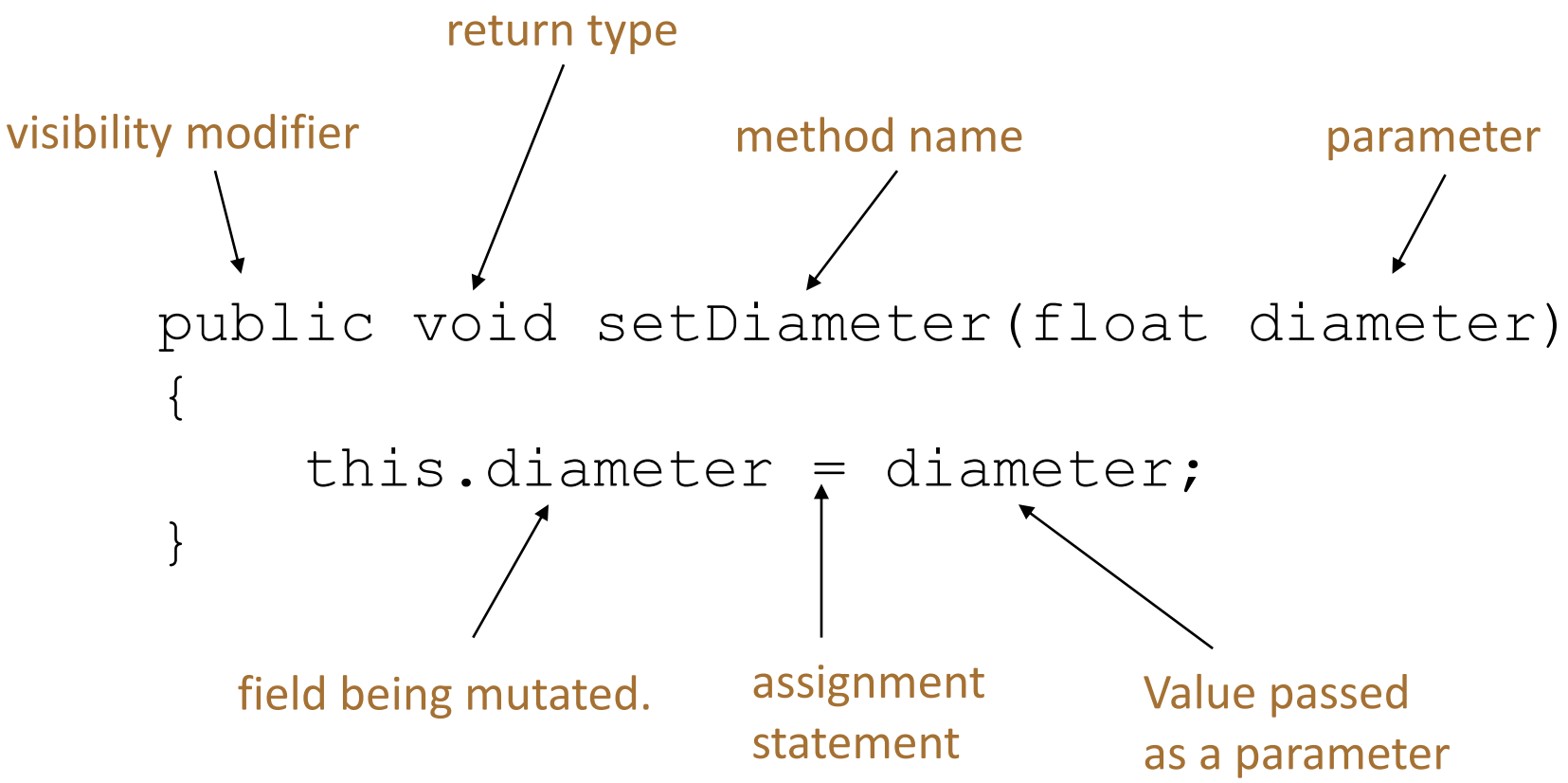
Mutator methods change (i.e. mutate!) an object's state.

A 'setter' method is a specific type of mutator method and typically:

- contains an assignment statement
- takes in a parameter
- changes the object state.

```
public class Spot
{
  private float xCoord, yCoord;
  private float diameter;

  //constructors…
  // getters…
 //     setters    //
public void setDiameter(float diameter){
    this.diameter = diameter;
  }

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

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

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

Improving the constructor with validation

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

```
public 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

```
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).

toString()

```
public class Spot
{
  private float xCoord, yCoord;
  private float diameter;

  //constructors…
  //getters…
  //setters…

  public String toString()
  {
     return “(“ + xCoord + “,” + yCoord + “).  Diameter is:  “ + diameter;
  }
}
```

toString()

- The toString() method returns a string version of an object.
- This is a useful method and you will write a toString() method for most of your classes.
- When you print an object, Java automatically calls the toString() method e.g.

```
Spot spot = new Spot();

//both of these lines of code do the same thing
System.out.println(spot);
System.out.println(spot.toString());
```

Other Methods

Spot
xCoord *yCoord* *diameter*
Spot() *Spot(float, float, float)* ***display()*** *getXCoord()* *getYCoord()* *getDiameter()* *setXCoord(float)* *setYCoord(float)* *setDiameter(float)* *toString()*

```
public class Spot
{
  private float xCoord, yCoord;
  private float diameter;

  //constructors…
  //getters…
  //setters…

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

}
```

Using the Spot Class

Declaring an object **sp**, of type **Spot**.

```
Spot sp;
```

Calling the **Spot(float, float, float)** *constructor* to build the **sp** object in memory.

```
sp = new Spot(40,30,45);
```

Calling the display method, over the **sp** object.

```
sp.display();
```

Questions?