More on Classes

Adding behaviour

Produced by: Dr. Siobhán Drohan
Mr. Colm Dunphy
Mr. Diarmuid O'Connor

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

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

Topics list

1. Recap: **Classes and Objects**

2. Recap on the **Spot class**:
   - v1.0 (**default constructor**)
   - v2.0 (**constructor with parameters**)
   - v3.0 (**overloading constructors**)

3. Adding **behaviours** to the Spot class:
   - v4.0 (**display()**)
   - v5.x (**colour()**)
   - v6.0 (**move()**)
   - v6.1 (**this** keyword – name overloading)

Classes and Objects

- A **class** defines a group of related
  - **fields** (variables, properties, attributes)
  - **methods** (functions – that manipulate those fields)
- An **object** is a single **instance** of a class
  - i.e. an object is created from a class.
- Many **objects** can be constructed from a single **class** definition.
- Analogy
  - Cake
    - A **class** is like a recipe for a cake.
    - An **object** is the cake baked from that recipe
    - You can bake lots of (cakes) **objects** from a single recipe

Class Names

- should match its purpose.
- any word or words.
- begin with a **Capital letter** and not be pluralised.
  - E.g. **S**pot
  - E.g. **A**pple

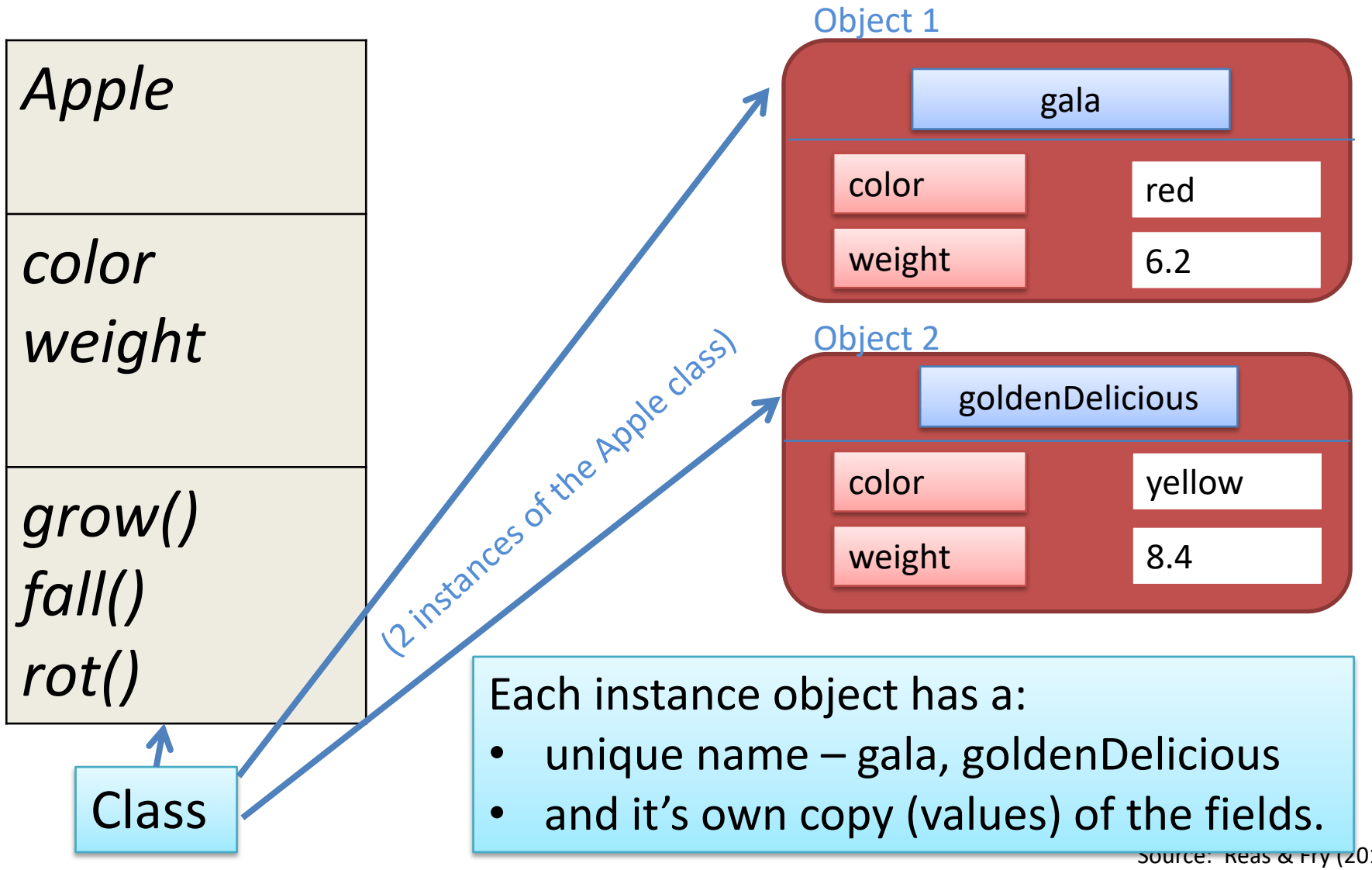
Source: Reas & Fry (2014)

Object example: Apple

Object Name	Apple
Fields (variables, properties, attributes)	color weight
Methods (functions)	grow() fall() rot()

Source: Reas & Fry (2014)

Apple Object(s)



Each instance object has a:

- unique name – gala, goldenDelicious
- and it's own copy (values) of the fields.

Source: Reas & Fry (2014)

Using an Object's **fields** and **methods**

- The *fields* and *methods* of an object are accessed with the **dot operator** i.e. external calls.

object.property
object.method

FIELDS

gala**.color**	Gives access to the color value in the gala object.
goldenDelicious**.color**	Gives access to the color value in the goldenDelicious object.

METHODS

gala**.grow()**	Runs the grow() method inside the gala object.
goldenDelicious**.fall()**	Runs the fall() method inside the goldenDelicious object.

Source: Reas & Fry (2014)

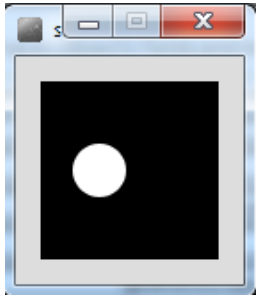
Topics list

1. Recap: Classes and Objects

2. Recap on the Spot class:
   - v1.0 (**default constructor**)
   - v2.0 (**constructor with parameters**)
   - v3.0 (**overloading constructors**)

3. Adding behaviours to the Spot class:
   - v4.0 (**display()**)
   - v5.x (**colour()**)
   - v6.0 (**move()**)
   - v6.1 (**this** keyword – name overloading)

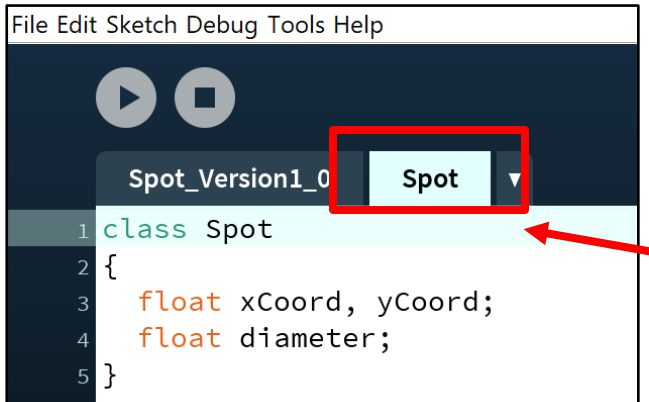
Spot Class – Version 1.0



Defining the **class**

```
class Spot
{
  float xCoord, yCoord;
  float diameter;
}
```

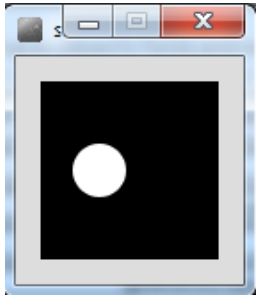
Declaring the **fields** in the class



In the PDE, place this code in a new **tab**, called Spot

Source: Reas & Fry (2014)

Spot Class – Version 1.0



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

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

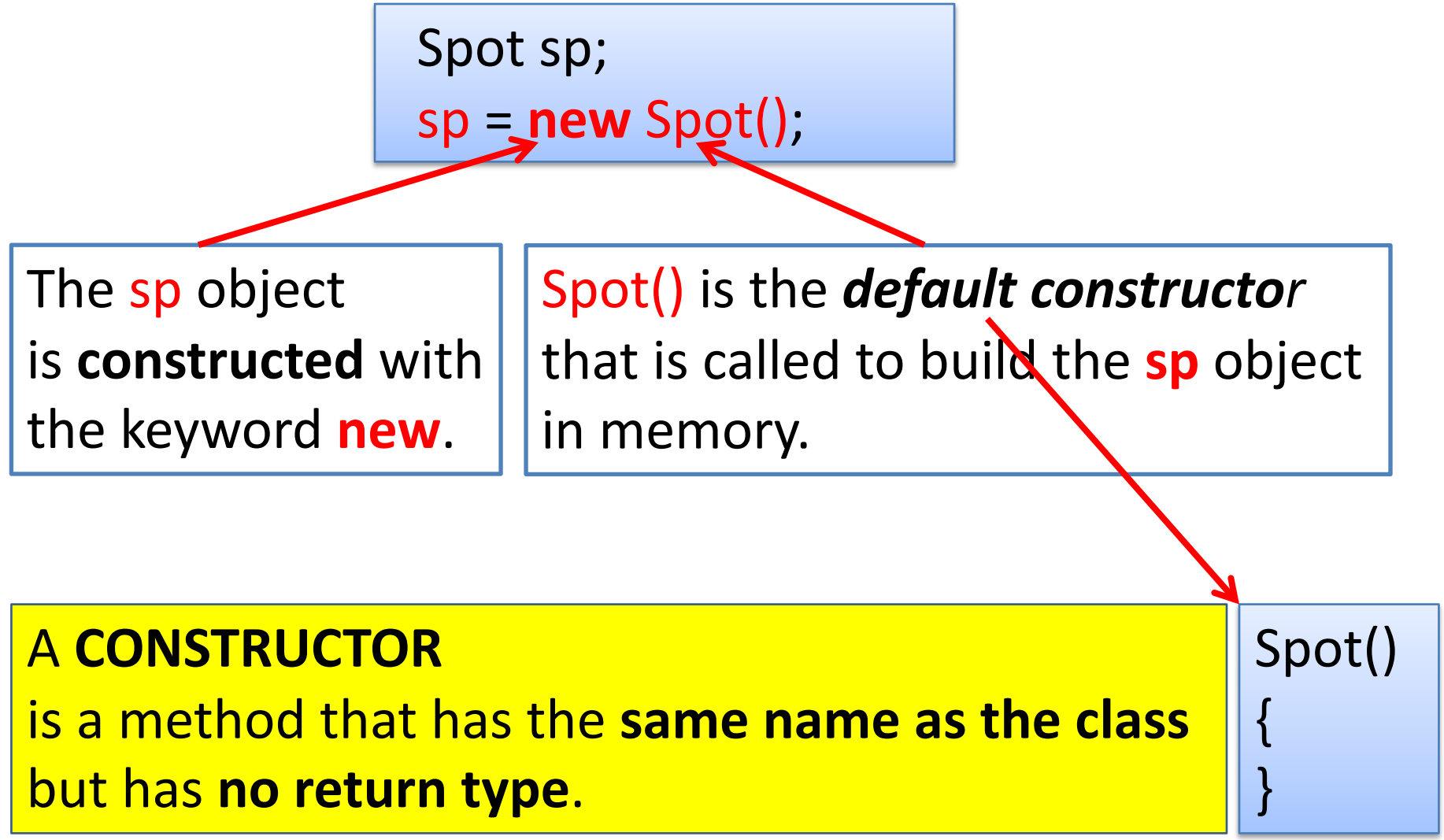
Initialising the fields in the **sp** object with a starting value.

Calling the ellipse method, using the fields in the **sp** object as arguments.

```
Spot sp;

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

void draw(){
  background(0);
  ellipse(sp.xCoord, sp.yCoord,
                  sp.diameter, sp.diameter);
}
```

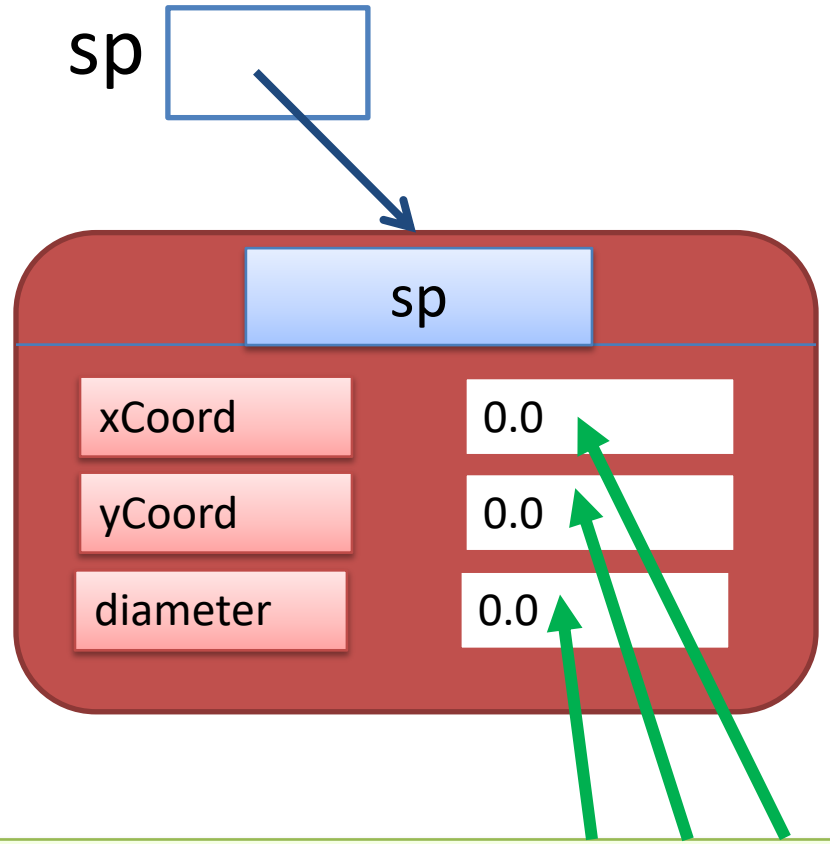
```
class Spot
{
  float xCoord, yCoord;
  float diameter;
}
```

Constructors



Default Constructor

```
class Spot
{
    float xCoord;
    float yCoord;
    float diameter;

     //Default Constructor
     Spot()
     {
     }
}
```



- The constructor stores initial values in the fields.

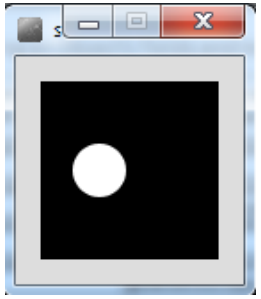
Topics list

1. Recap: Classes and Objects

2. Recap on the Spot class:
   - v1.0 (**default constructor**)
   - v2.0 (**constructor with parameters**)
   - v3.0 (**overloading constructors**)

3. Adding behaviours to the Spot class:
   - v4.0 (**display()**)
   - v5.x (**colour()**)
   - v6.0 (**move()**)
   - v6.1 (**this** keyword – name overloading)

Spot Class – Version 2.0



```
Spot sp;

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

void draw()
{
  background(0);
  ellipse(sp.xCoord, sp.yCoord, sp.diameter, sp.diameter);
}
```

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

  Spot (float xPos, float yPos, float diamtr)
  {
    xCoord = xPos;
    yCoord = yPos;
    diameter = diamtr;
  }
}
```

Source: Reas & Fry (2014)

Topics list

1. Recap: Classes and Objects

2. Recap on the Spot class:
   - v1.0 (**default constructor**)
   - v2.0 (**constructor with parameters**)
   - v3.0 (**overloading constructors**)

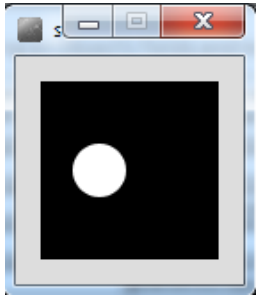
3. Adding behaviours to the Spot class:
   - v4.0 (**display()**)
   - v5.x (**colour()**)
   - v6.0 (**move()**)
   - v6.1 (**this** keyword – name overloading)

Overloading Constructors

- We can have as many constructors as our design requires, ONCE they have unique parameter lists.

- We are overloading our constructors in Version 3.0…

Spot Class – Version 3.0



overloading

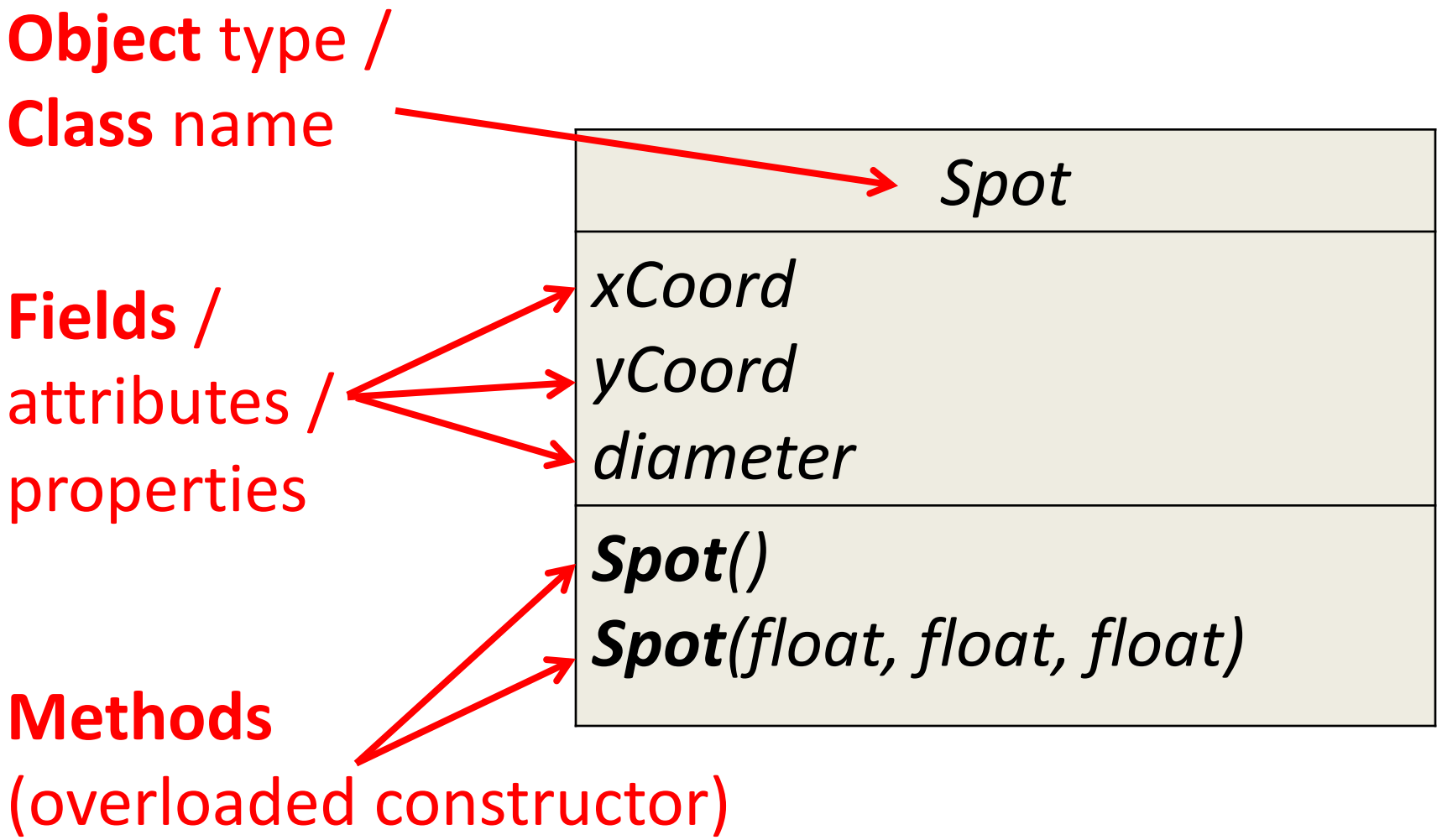
```
class Spot{
  float xCoord, yCoord;
  float diameter;

  Spot() {
  }

  Spot (float xPos, float yPos, float diamtr){
    xCoord = xPos;
    yCoord = yPos;
    diameter = diamtr;
  }
}
```

Default Constructor with NO parameters

A second **Constructor with a** (float, float, float) **parameter list**

Source: Reas & Fry (2014)

Class Diagram for Spot Version 3.0



Source: Reas & Fry (2014)

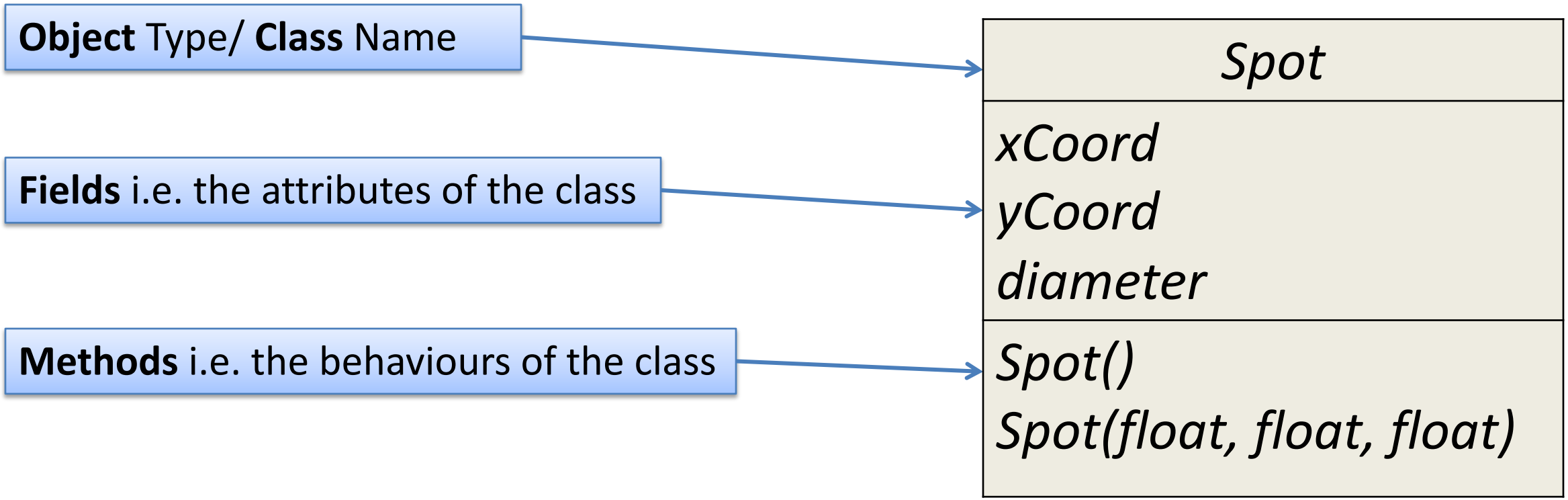
Topics list

1. Recap: Classes and Objects

2. Recap on the Spot class:
   - v1.0 (**default constructor**)
   - v2.0 (**constructor with parameters**)
   - v3.0 (**overloading constructors**)

3. Adding behaviours to the Spot class:
   - v4.0 (**display()**)
   - v5.x (**colour()**)
   - v6.0 (**move()**)
   - v6.1 (**this** keyword – name overloading)

Class Diagram for Spot Version 3.0



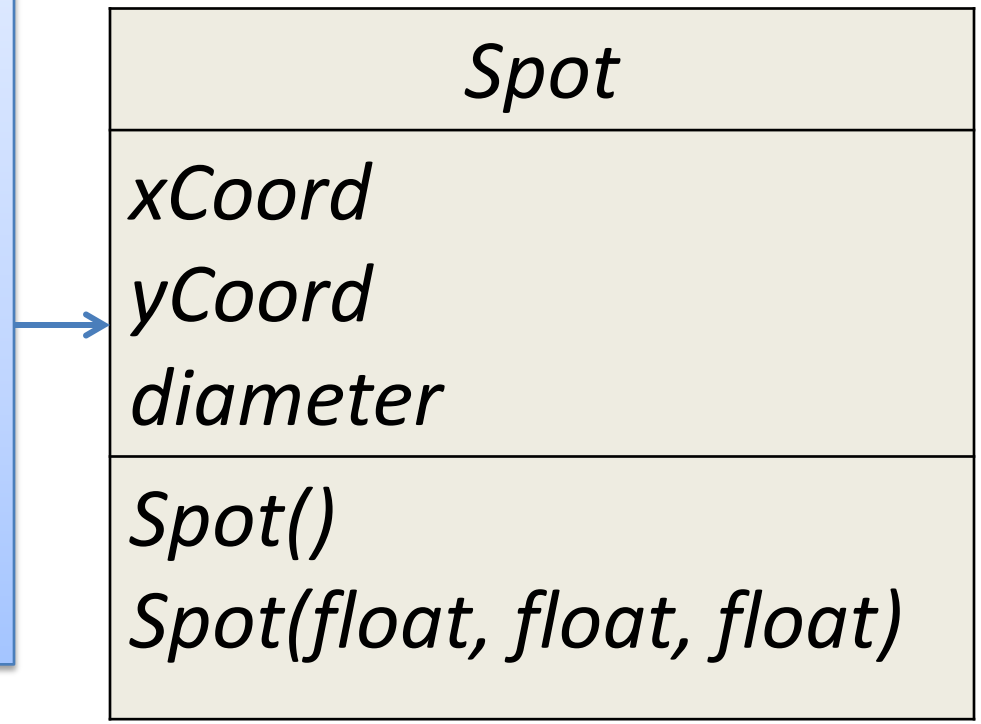
Source: Reas & Fry (2014)

Class Diagram for Spot Version 3.0

So far,
we only have overloaded constructors for our class
(they create the objects of our class).

We have not defined any **behaviours** for our class
e.g.
- **display** the spot,
- **colour** the spot,
- **move** the spot,
- and so on.

As it stands, the Spot class is not very useful!



Source: Reas & Fry (2014)

Spot – adding a "**display**" behaviour

- We want to add a behaviour to the Spot class that will draw the Spot on the screen.

- To add behaviour to a class, we write a **method** inside the class.

- We will call this method **display()**.

Source: Reas & Fry (2014)

display() method

- The method signature is:

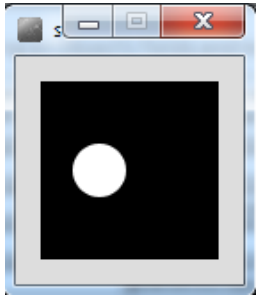
  void display()

- The method's job:
  - is to draw the spot on the display window using the values stored in the attributes (xCoord, yCoord, diameter).

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

Source: Reas & Fry (2014)

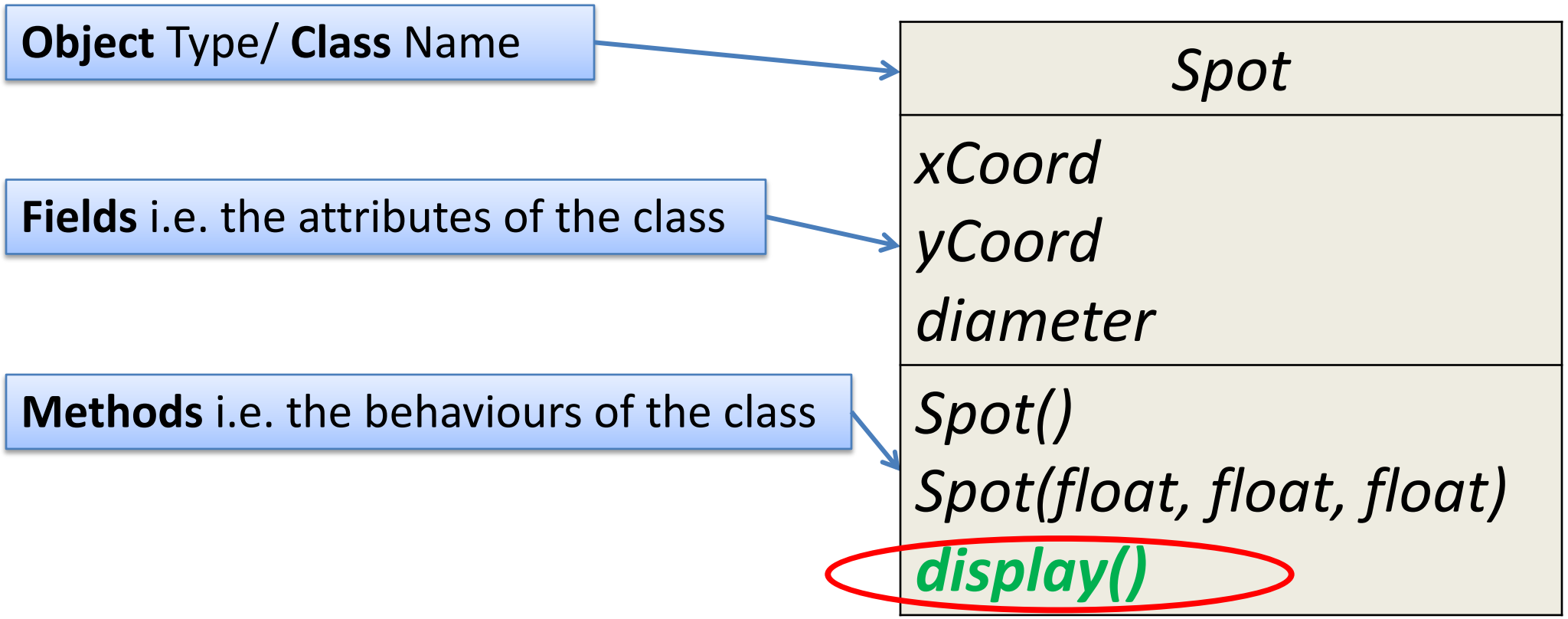
Spot Class – Version 4.0



```
Spot sp;

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

void draw()
{
  background(0);
  sp.display();
}
```

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

  Spot(){
  }

 Spot(float xPos, float yPos, float diamtr){
    xCoord = xPos;
    yCoord = yPos;
    diameter = diamtr;
  }

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

Class Diagram for Spot Version 4.0



Source: Reas & Fry (2014)

Topics list

1. ***Recap***: Classes and Objects

2. ***Recap*** on the Spot class:
   - v1.0 (**default constructor**)
   - v2.0 (**constructor with parameters**)
   - v3.0 (**overloading constructors**)

3. Adding behaviours to the Spot class:
   - v4.0 (**display()**)
   - v5.x (**colour()**)
   - v6.0 (**move()**)
   - v6.1 (**this** keyword – name overloading)

Spot – adding **RGB** "**colour**" behaviour

- We now want to add a behaviour to the Spot class that will colour the Spot, using RGB values on the screen.

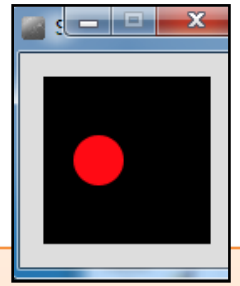
- We will need three extra attributes (**fields** / variables):

  int red
  int green
  int blue

- We will need to take in values for the red, green and blue fields using the parameters of our new **method**

  ***colour** (int redVal, int greenVal, int blueVal)*

Spot Class – Version 5.0



```
Spot sp;

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

void draw()
{
  background(0);
  sp.colour(255,10,20);
  sp.display();
}
```

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

  // constructors…

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

  void colour(int redVal, int greenVal, int blueVal){
    red = redVal;
    green = greenVal;
    blue = blueVal;
    fill (red, green, blue);
  }
}
```

New fields

New method

Spot – **Grayscale**“colour” behaviour

- We now want to add a behaviour to the Spot class
  that will colour the Spot, using a **Grayscale** value on the screen.

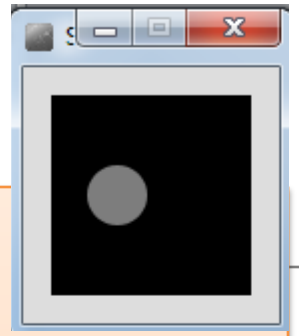
- To add this behaviour,
  we will need one extra attribute (**field** / variable):

  int gray

- We will need to take in a value for the gray field
  using the parameters of our new **method** e.g.:

  colour *(int grayVal)*

Spot Class – Version 5.1



```
Spot sp;

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

void draw()
{
  background(0);
  sp.colour(125);
  sp.display();
}
```

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

  // constructors…
  //display method…
  void colour(int redVal, int greenVal, int blueVal){
    red = redVal;
    green = greenVal;
    blue = blueVal;
    fill (red, green, blue);
  }

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

New field

New method

Spot – two colour behaviours

- We have **overloaded** the colour() method

  i.e.

  we have **two methods called colour()**
  that have different parameter lists:

  colour *(int redVal, int greenVal, int blueVal)*
  colour *(int grayVal)*

- Java knows which method to call
  based on matching the arguments passed to the method call.

Spot – two colour behaviours

Example Call 1

```
void draw()
{
  background(0);
  sp.colour(255,10,20);
  sp.display();
}
```

Example Call 2

```
void draw()
{
  background(0);
  sp.colour(125);
  sp.display();
}
```

```
class Spot{
  //variables…
  // constructors…
  //display method…
  void colour(int redVal, int greenVal, int blueVal){
    red = redVal;
    green = greenVal;
    blue = blueVal;
    fill (red, green, blue);
  }

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

Class Diagram for Spot Version 5.1

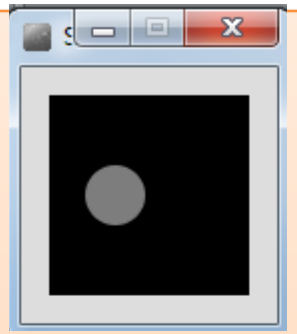
We have two constructors in our class.

But these constructors do not initialise our new fields, red, green, blue or gray.

Two new constructors are needed to initialise the Spot object to a starting:

- gray colour.
- RGB colour.

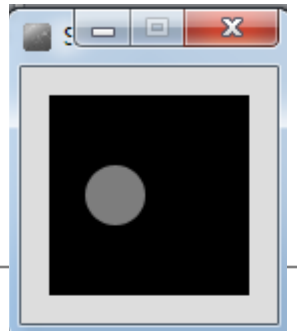
```
class Spot{
  // variables…
  // other constructors…
  Spot(float xPos, float yPos, float diamtr, int grayVal){
    xCoord = xPos;
    yCoord = yPos;
    diameter = diamtr;
    colour(grayVal);
  }

  Spot(float xPos, float yPos, float diamtr, int redVal, int greenVal, int blueVal){
    xCoord = xPos;
    yCoord = yPos;
    diameter = diamtr;
    colour(redVal, greenVal, blueVal);
  }
  // display method…
  // colour methods…
}
```



Spot Class – Version 5.2

Source: Reas & Fry (2014)

Using the "**GrayScale**" constructor

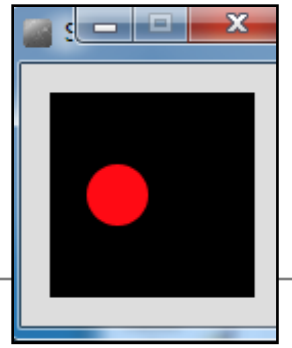


```
Spot sp;

void setup(){
  size (100,100);
  noStroke();
  sp = new Spot(33, 50, 30, 125);
}

void draw(){
  background(0);
  sp.display();
}
```

Spot Class – Version 5.2

Source: Reas & Fry (2014)

Using the "**RGB**" constructor

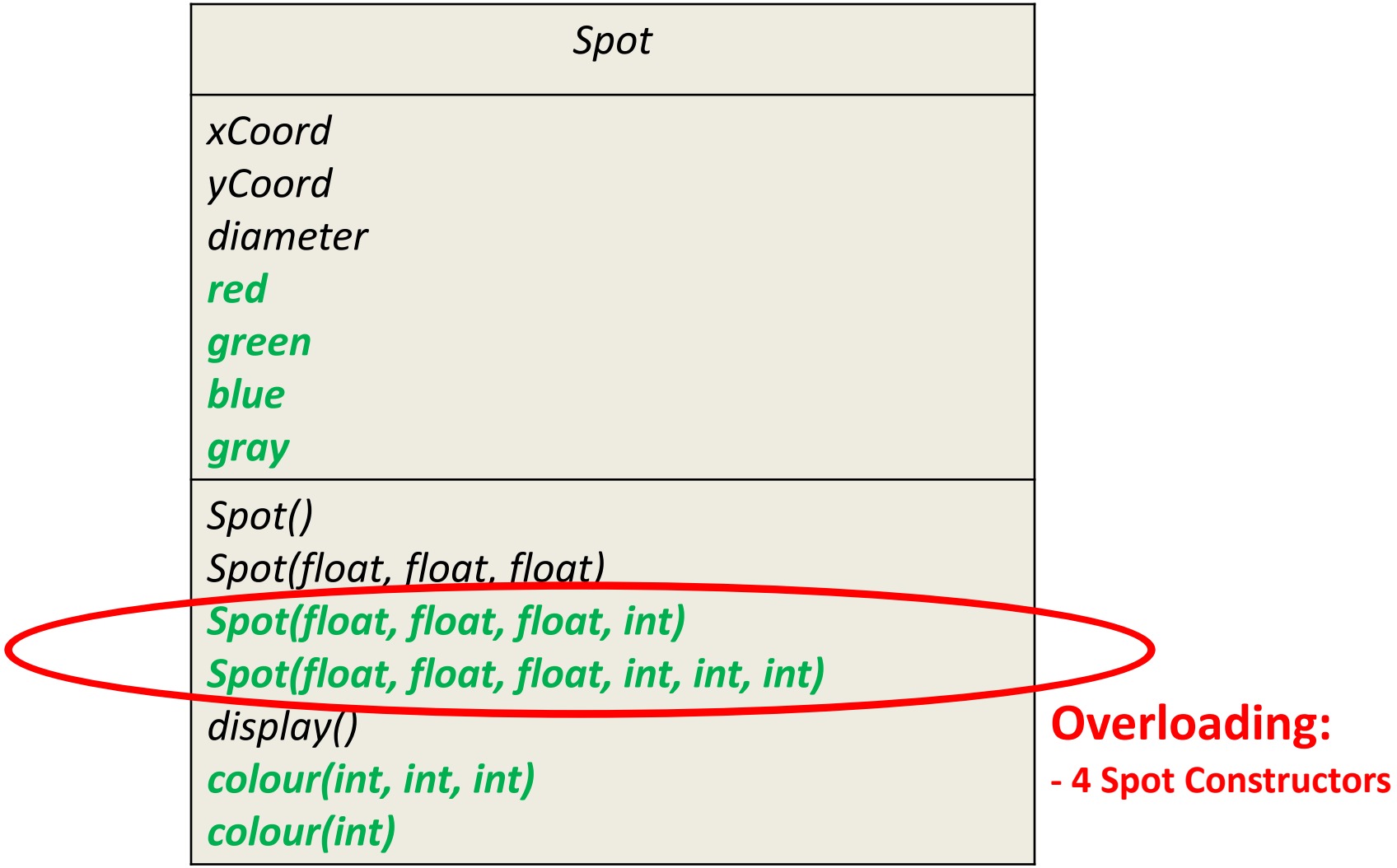


```
Spot sp;

void setup(){
  size (100,100);
  noStroke();
  sp = new Spot(33, 50, 30, 255,10,20);
}

void draw(){
  background(0);
  sp.display();
}
```

Spot Class – Version 5.2

Source: Reas & Fry (2014)

Class Diagram for Spot Version 5.2



Topics list

1. Recap: Classes and Objects

2. Recap on the Spot class:
   - v1.0 (**default constructor**)
   - v2.0 (**constructor with parameters**)
   - v3.0 (**overloading constructors**)

3. Adding behaviours to the Spot class:
   - v4.0 (**display()**)
   - v5.x (**colour()**)
   - v6.0 (**move()**)
   - v6.1 (**this** keyword – name overloading)

Spot – adding a “move” behaviour

- We now want to add a behaviour to the Spot class
  that will move the Spot around the screen.

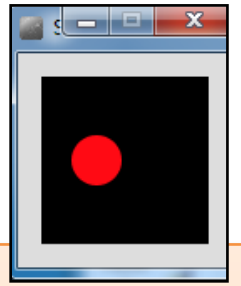
- To add this behaviour,
  we don’t need any extra attributes (fields / variables) as we already store the coordinates of the Spot:

  float xCoord
  float yCoord

- We will need to take in values
  for the **new position** of the Spot e.g.

  move *(float* ***xPos****, float* ***yPos****)*

Spot Class – Version 6.0

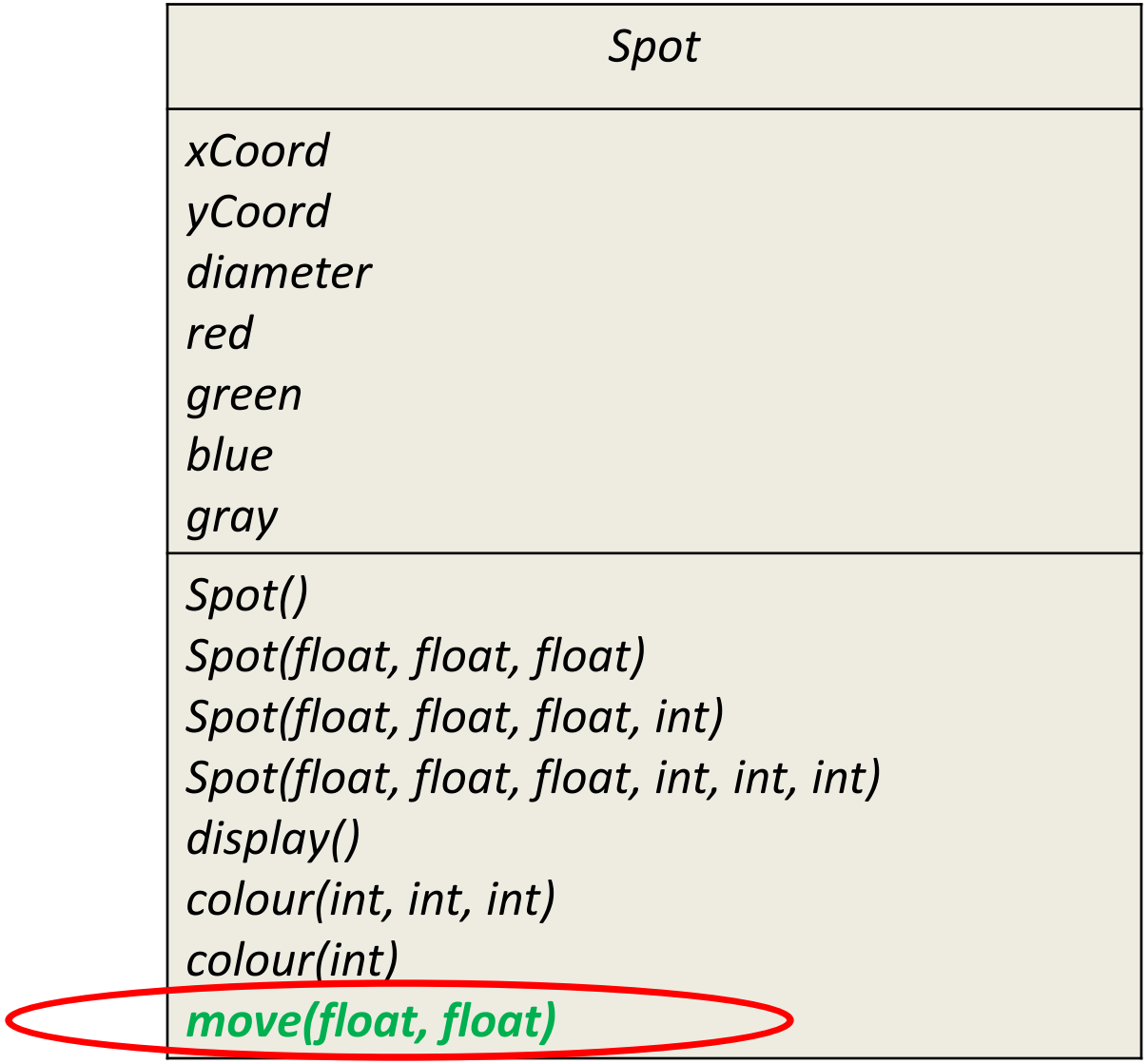


```
Spot sp;

void setup(){
  size (100,100);
  noStroke();
  sp = new Spot(33, 50, 30, 255,10,20);
}

void draw(){
  background(0);
  sp.display();
  sp.move (mouseX, mouseY);
}
```

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

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

  void move (float xPos, float yPos)
   {
     xCoord = xPos;
     yCoord = yPos;
   }
}
```

Source: Reas & Fry (2014)

Class Diagram for Spot Version 6.0



Topics list

1. Recap: Classes and Objects

2. Recap on the Spot class:
   - v1.0 (**default constructor**)
   - v2.0 (**constructor with parameters**)
   - v3.0 (**overloading constructors**)

3. Adding behaviours to the Spot class:
   - v4.0 (**display()**)
   - v5.x (**colour()**)
   - v6.0 (**move()**)
   - v6.1 (**this** keyword – name overloading)

this keyword

- The class Spot contains many fields
  e.g.:
  - xCoord, yCoord, diameter

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

  Spot(float xPos, float yPos, float diamtr)
  {
     xCoord = xPos;
     yCoord = yPos;
     diameter = diamtr;
  }
```

this keyword

- The class Spot contains many fields e.g.:
  - xCoord, yCoord, diameter
- One of the Spot constructors takes three parameters:
  - xPos, yPos, diamtr

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

  Spot (float xPos, float yPos, float diamtr)
  {
     xCoord = xPos;
     yCoord = yPos;
     diameter = diamtr;
  }
```

this keyword

- It would be nice to name the parameters passed into the Spot constructor the **same names as the instance fields**.

- This is called **name overloading**.

- But how will Java know
  which variable we are referring to?

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

  Spot (float xPos, float yPos, float diamtr)
  {
     xCoord = xPos;
     yCoord = yPos;
     diameter = diamtr;
  }
```

this keyword

We can use the this keyword to distinguish between them

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

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

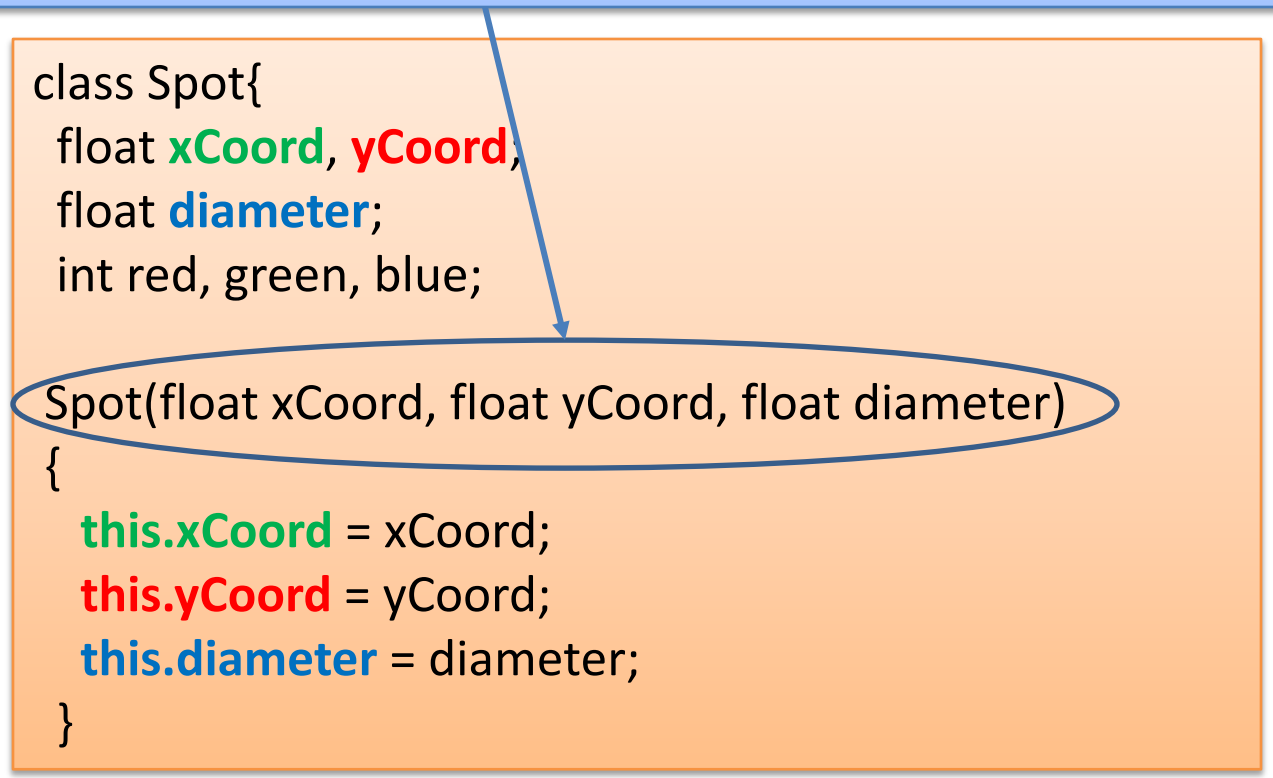
this keyword

this refers to the current object fields.

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

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

this keyword

These are local fields that are destroyed
as soon as the Spot constructor finishes executing.



this keyword – other examples

```
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);
 }
```

To clairfy, in the statement:

this.x = x;

Where **this.x** refers to the object's property / field

and **x** on it's own
is the parameter passed into the method

substitute x for any property/field

This describes **NAME OVERLOADING**

Summary

1. Recap: Classes and Objects

2. Recap on the Spot class:
   - v1.0 (**default constructor**)
   - v2.0 (**constructor with parameters**)
   - v3.0 (**overloading constructors**)

3. Adding behaviours to the Spot class:
   - v4.0 (**display()**)
   - v5.x (**colour()**)
   - v6.0 (**move()**)
   - v6.1 (**this** keyword – **name overloading**)

Questions?

Poll – Your computer

References

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