Classes and Objects

An introduction

Produced by: Dr. Siobhán Drohan
Mairead Meagher

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.

- A Building Analogy:
  - A class is like a blueprint for a building and an object is a building constructed from that blueprint.
- A Cake Analogy:
  - A class is like a recipe for a cake and an object is the cake.

Source: Reas & Fry (2014)

Many objects

- Many objects can be constructed from a single class definition.
- Each object must have a unique name within the program.

- A Building Analogy:
  - With a building blueprint (class), many buildings (objects) can be built from it.
- A Cake Analogy:
  - With a cake recipe (class), many cakes (objects) can be made from it.

Source: Reas & Fry (2014)

Methods (functions) and Fields (variables)

- In object-oriented programming (e.g. Java), you create objects by grouping together related methods (functions) and fields (variables).
- Objects can be related to real-world artefacts.

Source: Reas & Fry (2014)

Object example: Apple

Object Name	Apple
Fields (variables)	color, weight

Methods (functions)	grow()
	fall()
	rot()

Source: Reas & Fry (2014)

Object example: Butterfly

Object Name	Butterfly
Fields (variables)	species, gender

Methods (functions)	flapWings()
	land()

Source: Reas & Fry (2014)

Object example: Radio

Object Name	Radio
Fields (variables)	frequency, volume

Methods (functions)	turnOn()
	tune()
	setVolume()

Source: Reas & Fry (2014)

Object example: Car

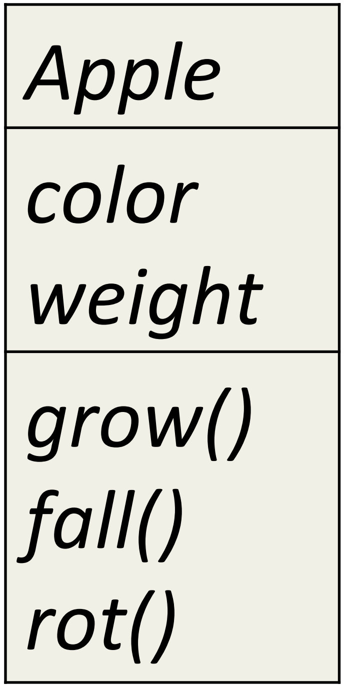
Object Name	Car
Fields (variables)	make, model, color, year

Methods (functions)	accelerate()
	brake()
	turn()

Source: Reas & Fry (2014)

Apple Class

- To make a software simulation of an Apple:
  - The **grow()** method might have inputs for temperature and moisture. The **grow(**) method can increase the weight field of the apple based on these inputs.
  - The **fall()** method can continually check the weight and cause the apple to fall to the ground when the weight goes above a threshold.
  - The **rot()** method could then take over, beginning to decrease the value of the weight field and change the colour fields.



Apple

color
weight

grow()
fall()
rot()

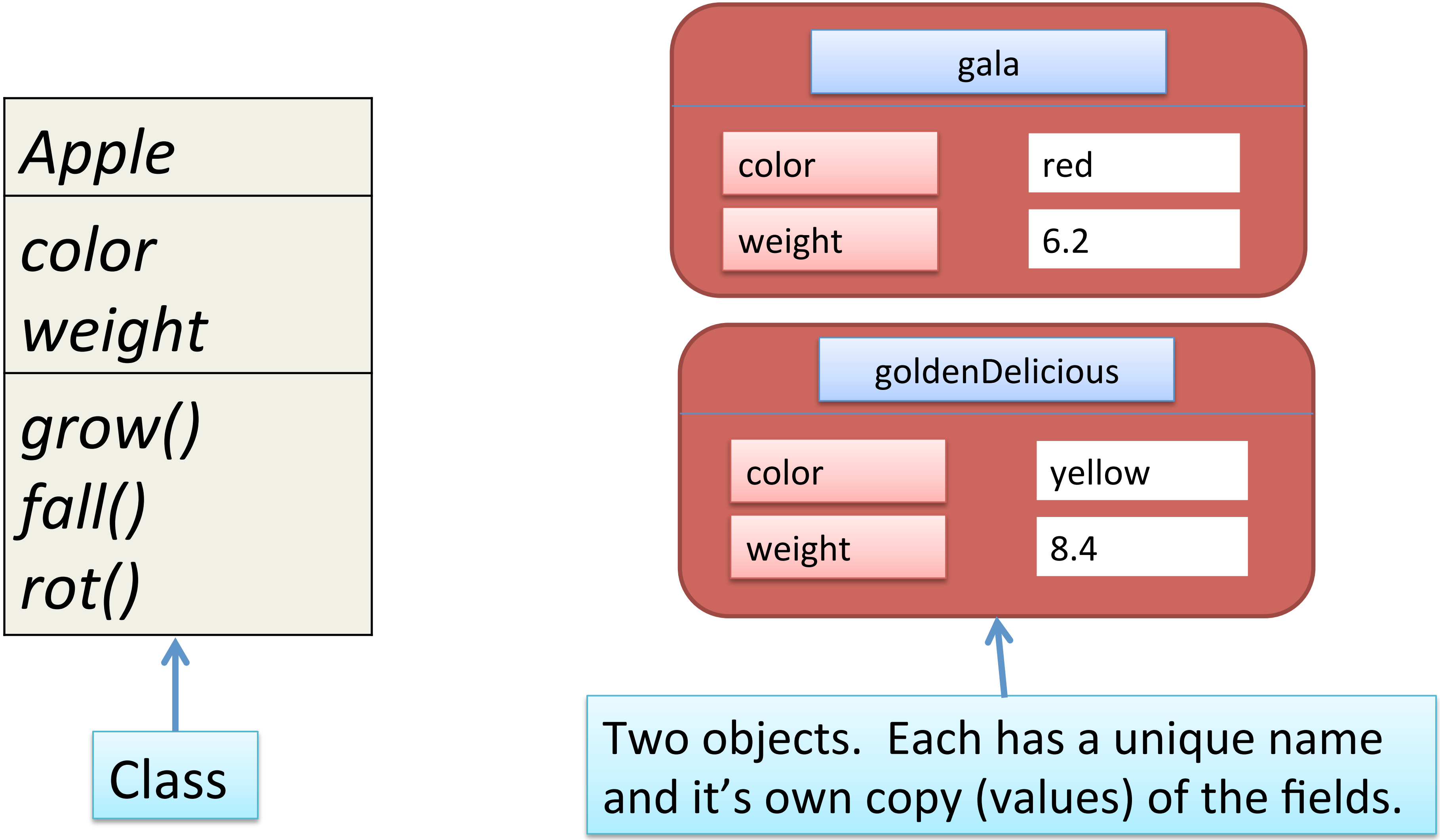
Source: Reas & Fry (2014)

Apple Object(s)

- We saw earlier that:
  - An object is created (instantiated) from a class.
  - A class can have many objects created from it.
  - Each object must have a unique name within the program.

Source: Reas & Fry (2014)

Apple Object(s)



Apple

color
weight

grow()
fall()
rot()

Class

gala	
color	red
weight	6.2

goldenDelicious	
color	yellow
weight	8.4

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

Source: Reas & Fry (2014)

Object State

There are two objects of type Apple.

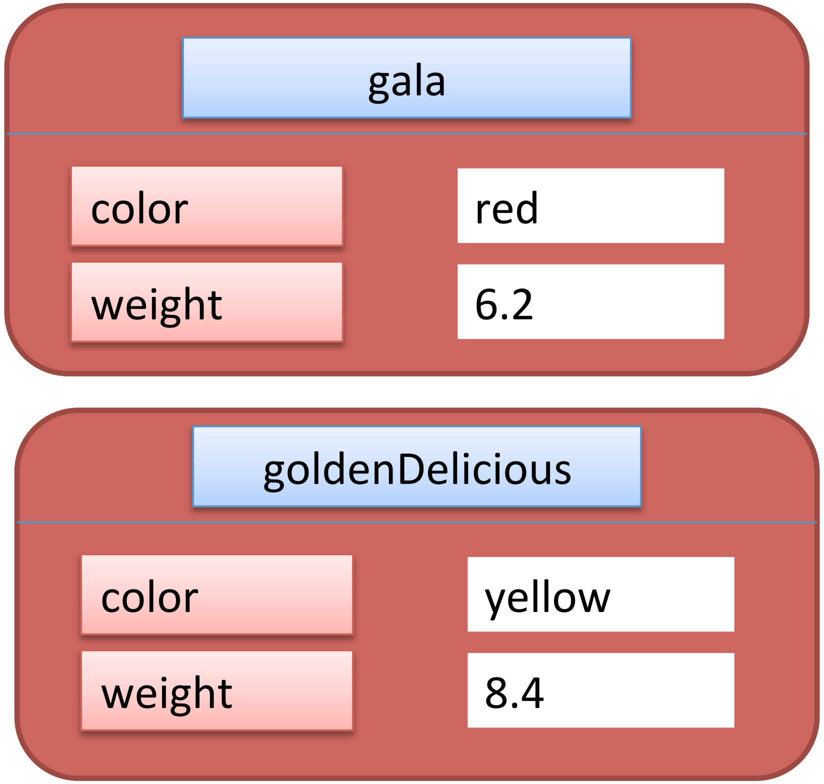
Each has a unique name:

  gala

  goldenDelicious

Each object has a different **object state**:

  each object has it's own copy of the fields (color and weight) in memory and has it's own data stored in these fields.



gala

color	red
weight	6.2

goldenDelicious

color	yellow
weight	8.4

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.

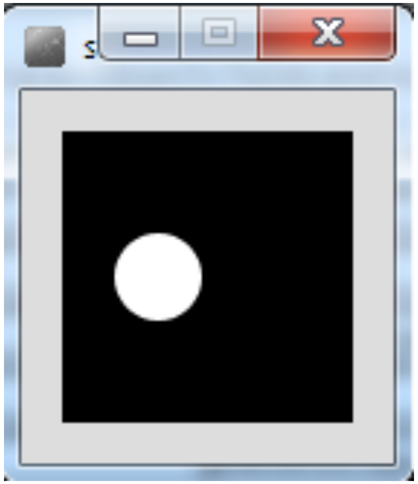
gala.color	Gives access to the color value in the gala object.
goldenDelicious.color	Gives access to the color value in the goldenDelicious object.
gala.grow()	Runs the grow() method inside the gala object.
goldenDelicious.fall()	Runs the fall() method inside the goldenDelicious object.

Source: Reas & Fry (2014)

Creating your first class

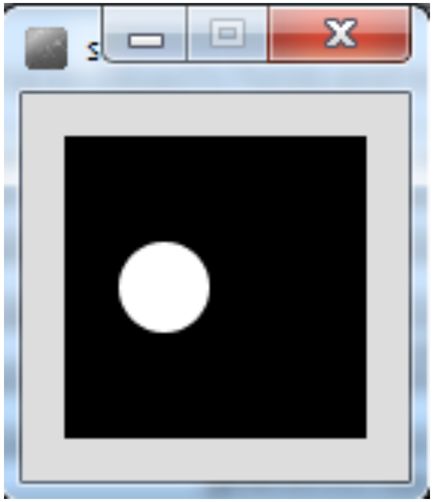
- We are going to start with sample code that draws a white spot on a black background.

- We will refactor this code by writing a class that will draw and format this spot.



Source: Reas & Fry (2014)

Sample Code



```
float xCoord = 33.0;
float yCoord = 50.0;
float diameter = 30.0;

void setup(){
  size (100,100);
  noStroke();
}

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

Source: Reas & Fry (2014)

Creating your first class

- A class creates a unique data type.
- When creating a class, think carefully about what you want the code to do.

- First, we will start by listing the required fields (variables) and figure out what type they should be.

Source: Reas & Fry (2014)

Creating your first class – identifying the fields

```
float xCoord = 33.0;
float yCoord = 50.0;
float diameter = 30.0;

void setup(){
  size (100,100);
  noStroke();
}

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

The required fields are:

float xCoord *(x-coordinate of spot)*

float yCoord *(y-coordinate of spot)*

float diameter *(diameter of the spot)*

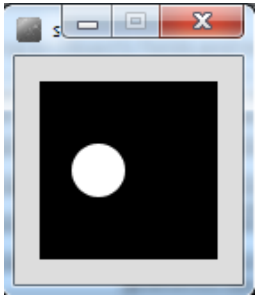
Source: Reas & Fry (2014)

Creating your first class – giving your class a name

- The name of a class should be carefully considered and should match its purpose.
- The name can be any word or words.
- It should begin with a capital letter and not be pluralised.
- For our first class, we could use names like:
  - Spot
  - Dot
  - Circle, etc.
- We will call our first class, **Spot**.

Source: Reas & Fry (2014)

Spot Class – Version 1.0

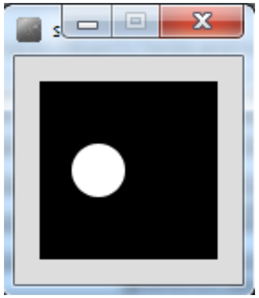


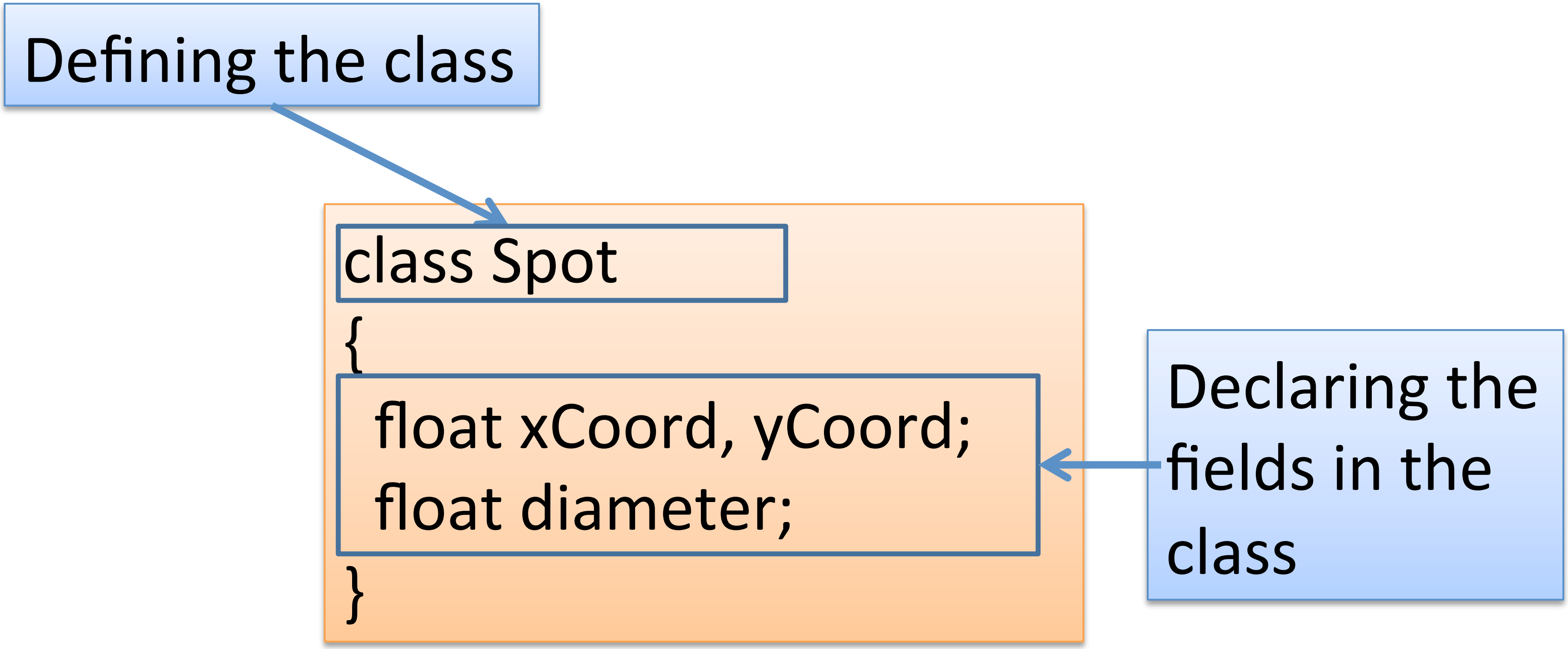
```
Spot sp;

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;
}
```

Source: Reas & Fry (2014)

Spot Class – Version 1.0

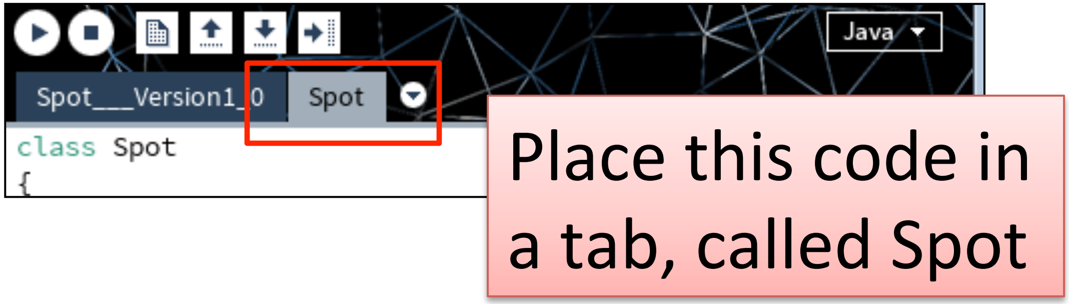




Defining the class

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

Declaring the fields in the class

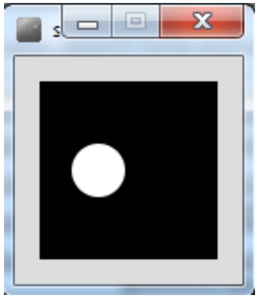


```
class Spot
{
```

Place this code in a 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.

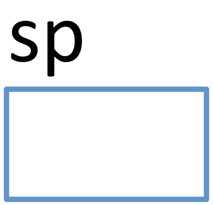
Initialising the fields in the **sp** object with a starting value.

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

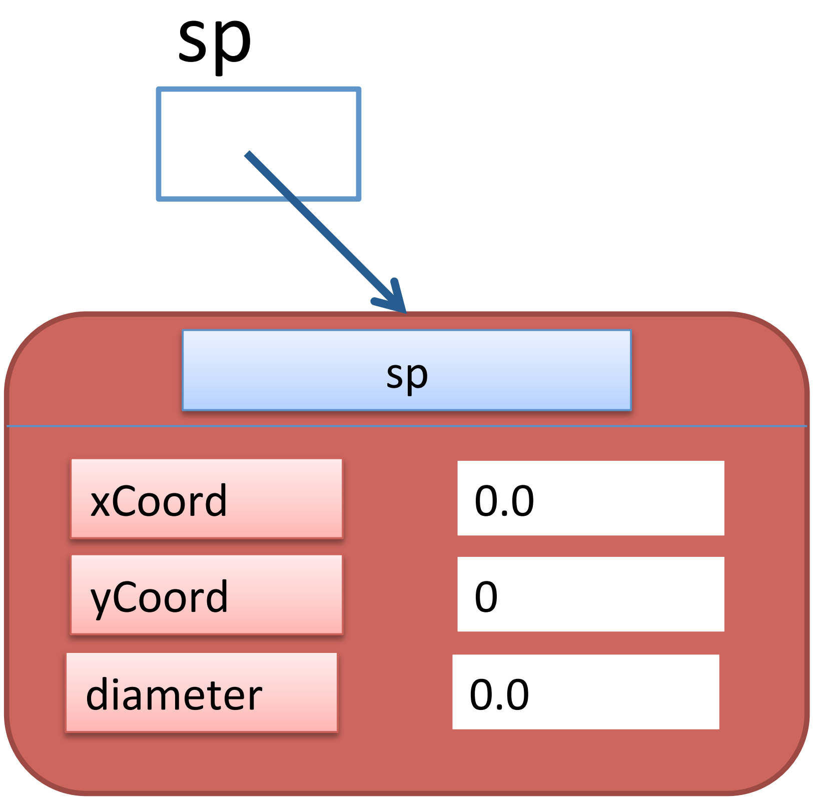
```
Spot sp;

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);
}
```

Constructors

```
Spot sp;
```

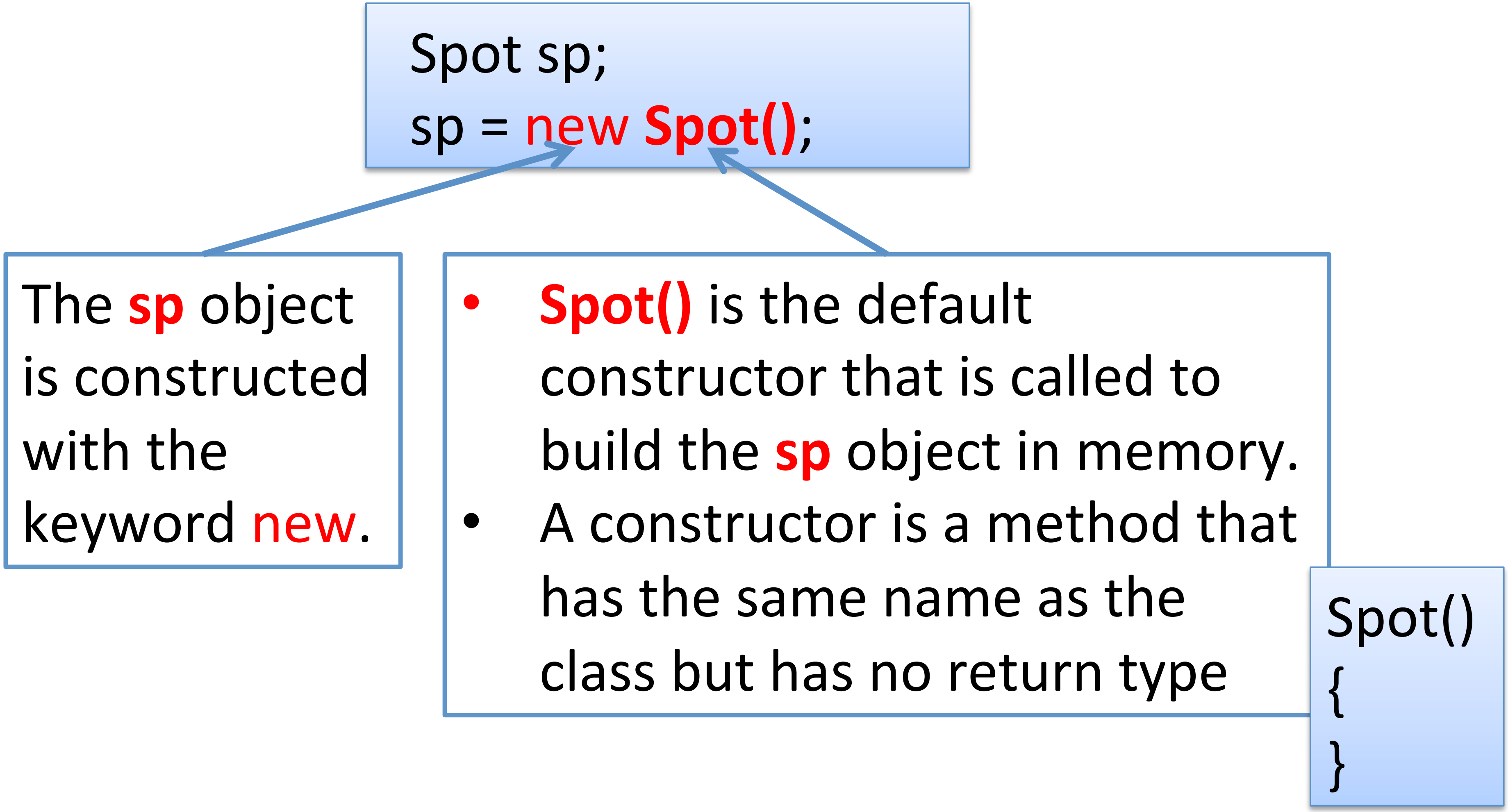


```
sp = new Spot();
```



xCoord	0.0
yCoord	0
diameter	0.0

Constructors



```
Spot sp;
sp = new Spot();
```

The **sp** object is constructed with the keyword new.

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

```
Spot()
{
}
```

Default Constructor

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

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

- The default constructor has an empty parameter list.
- If you don’t include a constructor in your class, the compiler inserts a default one for you (in the background…you won’t see it in your code).
- Here, the Spot() default constructor simply constructs the object.

Writing our first constructor

- The constructors can store initial values into the fields of the object.
- They often receive external parameter values for this.
- In this code, we initialised the xCoord, yCoord and diameter after calling the Spot() constructor.

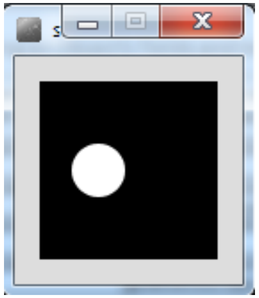
```
Spot sp;

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);
}
```

Writing our first constructor

- We want to write a new constructor that will take three parameters
  xPos,
  yPos
  diamtr
- These values will be used to initialise the
  xCoord,
  yCoord and
  diameter
  variables.

```
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);
}
```

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;
  }
}
```

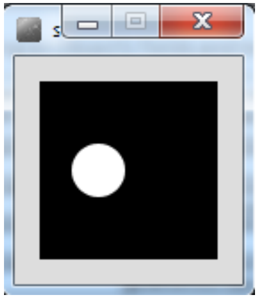
Source: Reas & Fry (2014)

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



```
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(){
  }

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

Source: Reas & Fry (2014)

Questions?

References

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