Classes and Objects in Java

Syeda Ajbina Nusrat, Lecturer, CSE, UITS

Contents

- Introduce to classes and objects in Java.
- Understand how some of the OO concepts learnt so far are supported in Java.
- Understand important features in Java classes.

Introduction

- Java is a true OO language and therefore the underlying structure of all Java programs is classes.
- Anything we wish to represent in Java must be encapsulated in a class that defines the “state” and “behaviour” of the basic program components known as objects.
- Classes create objects and objects use methods to communicate between them. They provide a convenient method for packaging a group of logically related data items and functions that work on them.
- A class essentially serves as a template for an object and behaves like a basic data type “int”. It is therefore important to understand how the fields and methods are defined in a class and how they are used to build a Java program that incorporates the basic OO concepts such as encapsulation, inheritance, and polymorphism.

- A class defines a new data type. This new data type is used to create objects of that type.
- A class is a template for an object and an object is an instance of a class.
- Class is user defined type.

Example:

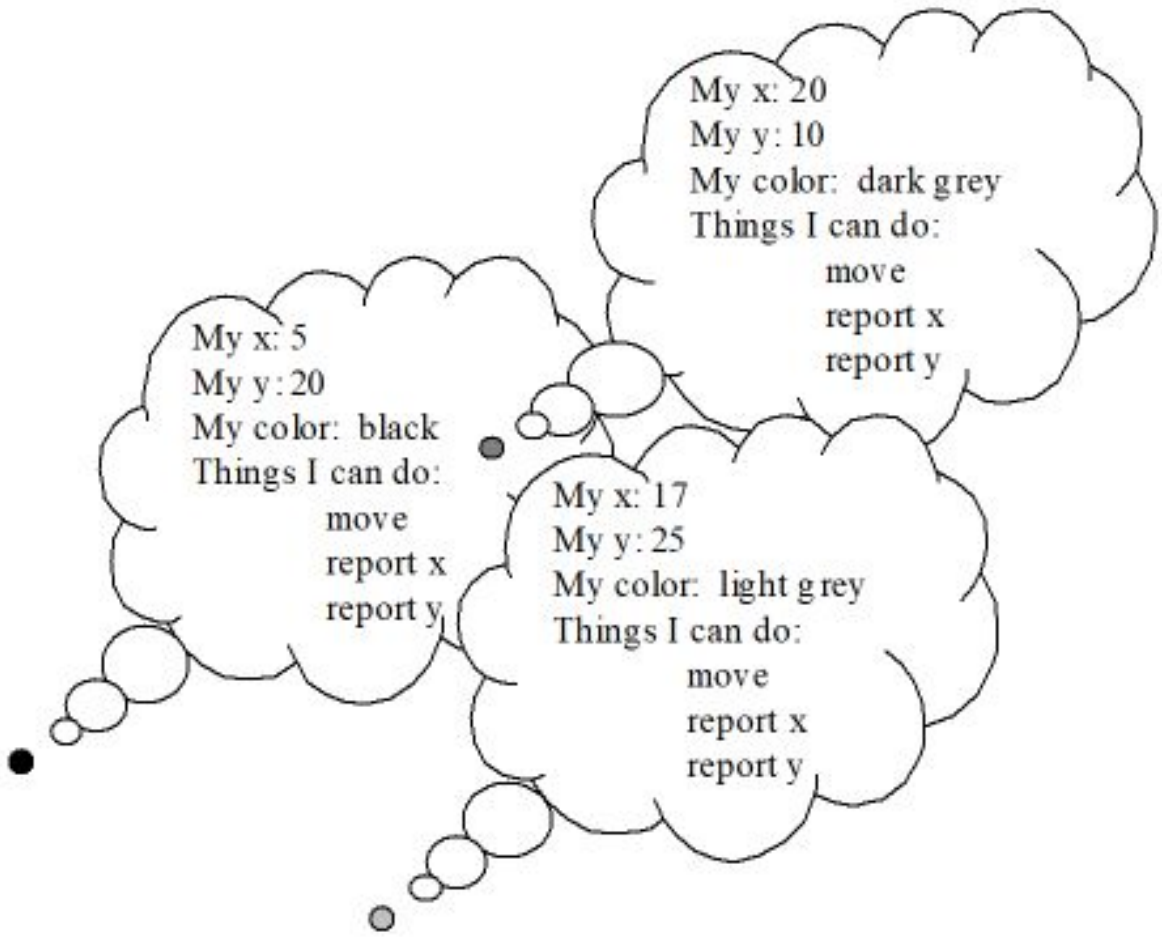
Colored points on the screen

What data goes into making one?

- Coordinates
- Color

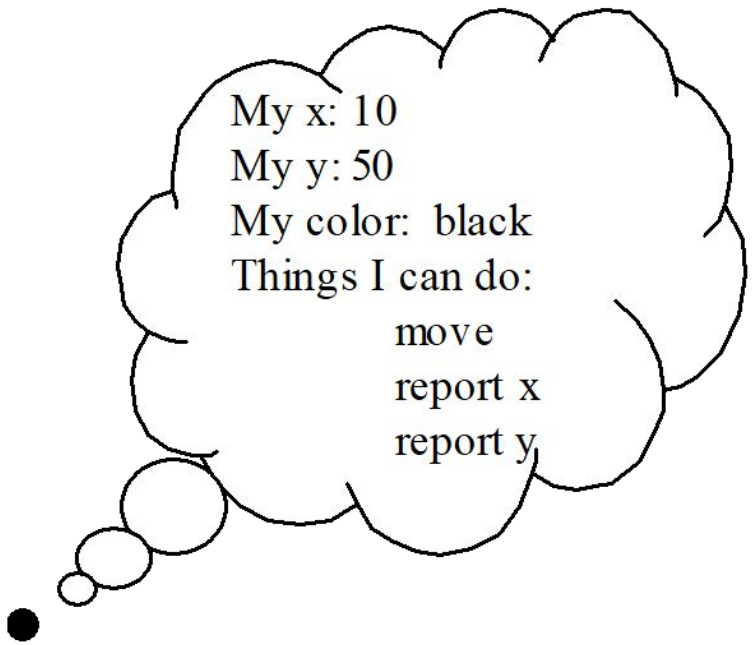
What should a point be able to do?

- Move itself
- Report its position


My x: 20
My y: 10
My color: dark grey
Things I can do:
move
report x
report y
My x: 5
My y: 20
My color: black
Things I can do:
move
report x
report y
My x: 17
My y: 25
My color: light grey
Things I can do:
move
report x
report y

JAVA Terminology

- ✓Each point is an *object*
- ✓Each includes three *data fields*
- ✓Each has three *methods*
- ✓Each is an *instance* of the same *class*

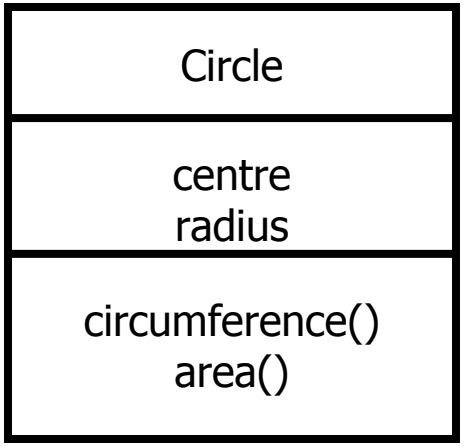


```
class Point {
  int xCoord;
  int yCoord;
  Point() {
        xCoord = 0;
        yCoord = 0;
  }
  int currentX() {
        return xCoord;
  }
  int currentY() {
        return yCoord;
  }
  void move (int newXCoord, int newYCoord) {

        xCoord = newXCoord;
        yCoord = newYCoord;

  }
}
```

```
class PointProcess{
 public static void main(String args[])
 {
    Point P=new Point();
    P.move(3,4);
    System.out.println("X is: "+P.currentX());
    System.out.println("Y is: "+Y.currentX());
 }
}
```

Classes

- A *class* is a collection of *fields* (data) and *methods* (procedure or function) that operate on that data.



Classes

- A *class* is a collection of *fields* (data) and *methods* (procedure or function) that operate on that data.
- The basic syntax for a class definition:

```
class  ClassName [extends
SuperClassName]
{
        [fields declaration]
        [methods declaration]
 }
```

- Bare bone class – no fields, no methods

```
public class Circle {
    // my circle class
}
```

Adding Fields: Class Circle with fields

- Add *fields*

```
public class Circle {
      public double x, y;  // centre coordinate
      public double r;     //  radius of the circle

}
```

- The fields (data) are also called the *instance* variables.

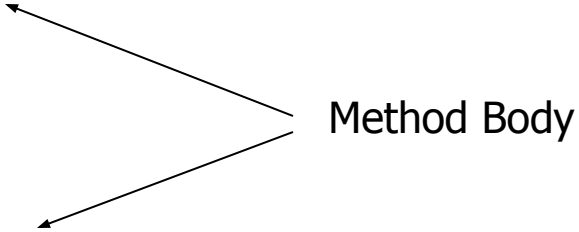
Adding Methods

- A class with only data fields has no life. Objects created by such a class cannot respond to any messages.
- Methods are declared inside the body of the class but immediately after the declaration of data fields.
- The general form of a method declaration is:

```
type MethodName (parameter-list)
{
        Method-body;
}
```

Adding Methods to Class Circle

```
public class Circle {

    public double x, y; // centre of the circle
    public double r;    // radius of circle

    //Methods to return circumference and area
    public double circumference() {
            return 2*3.14*r;
    }
    public double area() {
            return 3.14 * r * r;
    }
}
```



Data Abstraction

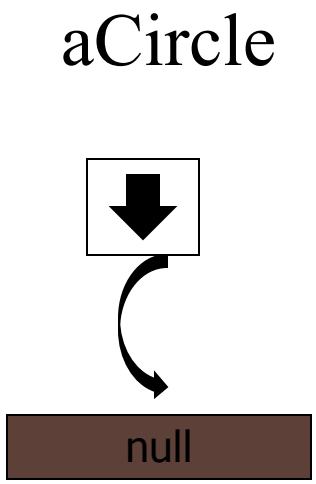
- Declare the Circle class, have created a new data type – Data Abstraction

- Can define variables (objects) of that type:

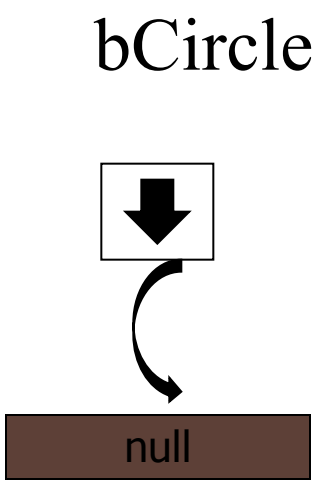
```
Circle  aCircle;
Circle  bCircle;
```

Class of Circle cont.

- aCircle, bCircle simply refers to a Circle object, not an object itself.



Points to nothing (Null Reference)

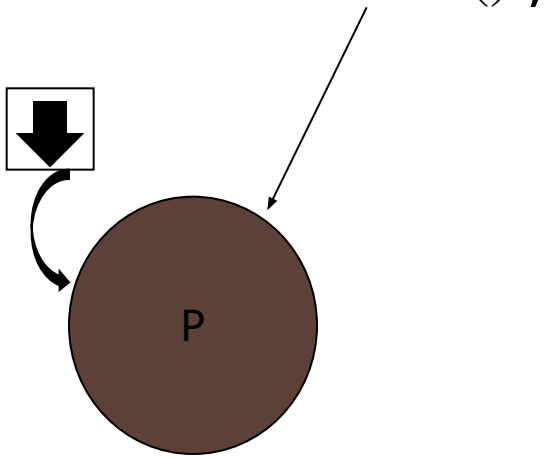


Points to nothing (Null Reference)

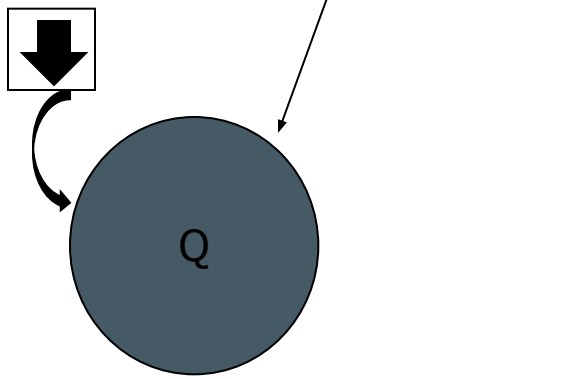
Creating objects of a class

- Objects are created dynamically using the *new* keyword.
- aCircle and bCircle refer to Circle objects

aCircle = new Circle() ;



bCircle = new Circle() ;



Creating objects of a class

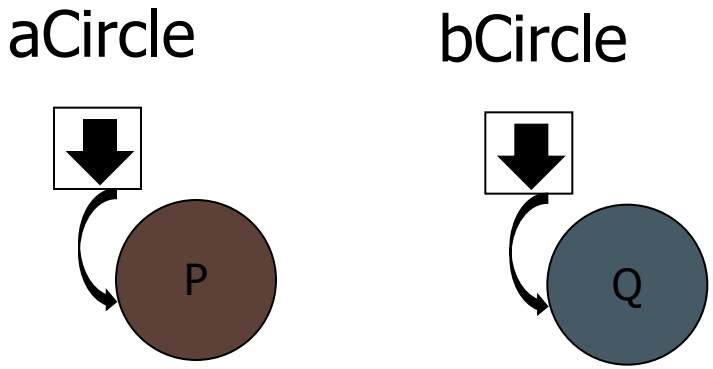
aCircle = new Circle();
bCircle = new Circle() ;

bCircle = aCircle;

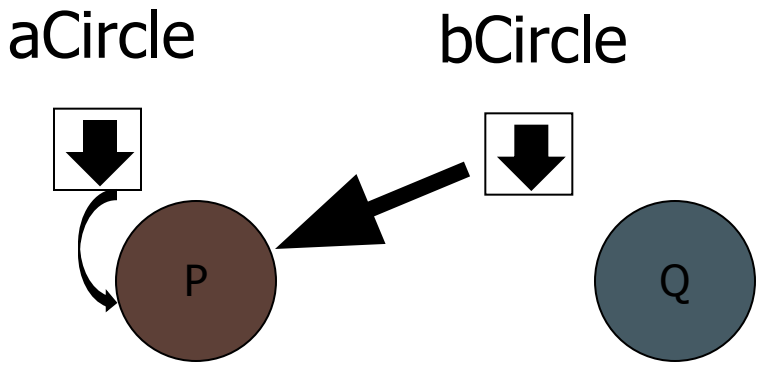
Creating objects of a class

```
aCircle = new Circle();
bCircle = new Circle() ;

bCircle = aCircle;
```

Before Assignment



After Assignment



Accessing Object/Circle Data

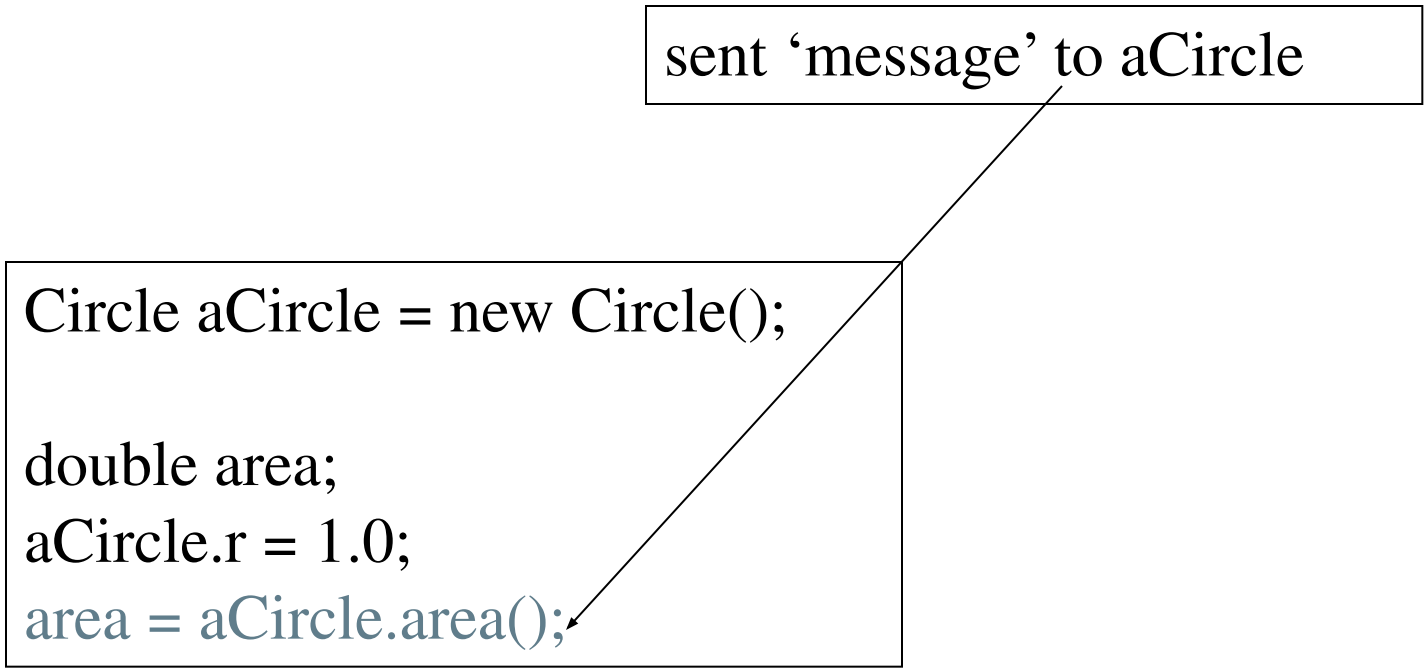
- Similar to C syntax for accessing data defined in a structure.

ObjectName.VariableName
ObjectName.MethodName(parameter-list)

```
Circle aCircle = new Circle();

aCircle.x = 2.0 // initialize center and radius
aCircle.y = 2.0
aCircle.r = 1.0
```

Executing Methods in Object/Circle

- Using Object Methods:



Using Circle Class

```
// Circle.java:  Contains both Circle class and its user class
//Add Circle class code here
class MyMain
{
	public static void main(String args[])
	{
		Circle aCircle;  // creating reference
		aCircle = new Circle(); // creating object
		aCircle.x = 10;  // assigning value to data field
		aCircle.y = 20;
		aCircle.r = 5;
		double area = aCircle.area(); // invoking method
		double circumf = aCircle.circumference();
		System.out.println("Radius="+aCircle.r+" Area="+area);
		System.out.println("Radius="+aCircle.r+" Circumference ="+circumf);
	}
}
```

```
Radius=5.0 Area=78.5
Radius=5.0 Circumference =31.400000000000002
```

References

- An reference variable holds the memory address.
- It is similar to the pointer.
- Key difference- cannot manipulate as pointer– It cannot point any arbitrary memory location or it cannot be manipulate like an integer.

The null reference

- An object reference variable that does not currently point to an object is called a *null reference*
- The reserved word null can be used to explicitly set a null reference:

```
name = null;
```

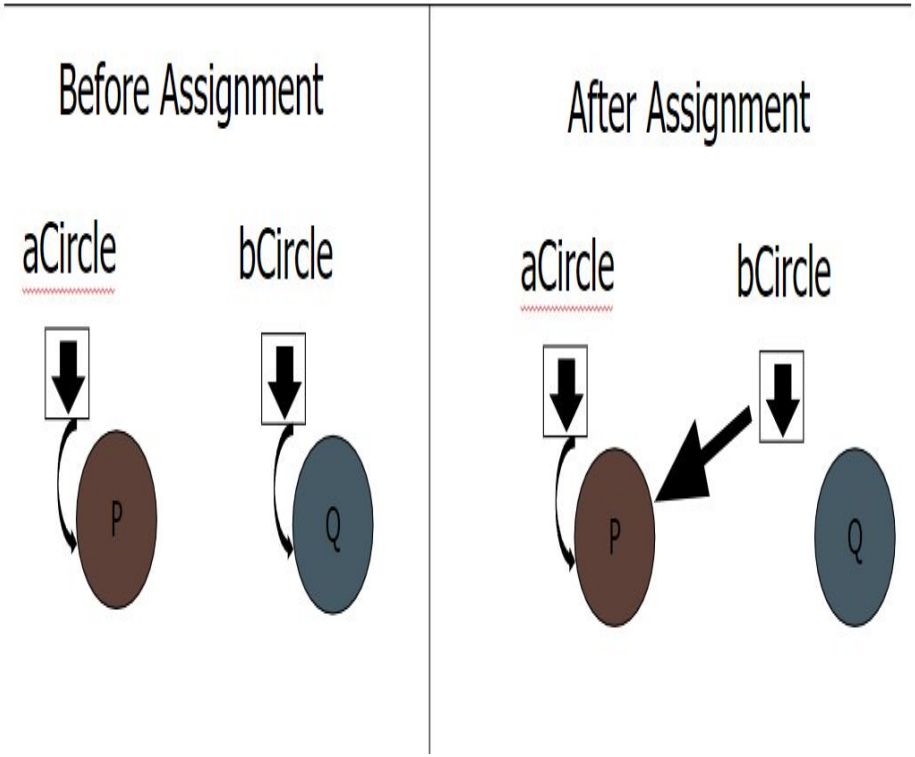
- An object reference variable declared at the class level is automatically initialized to null
- The programmer must carefully ensure that an object reference variable refers to a valid object before it is used

Automatic garbage collection

- Java handles deallocation automatically.
- Garbage collection system automatically deallocates space for the programmer.
- It runs periodically.
- It automatically releases space for an object, when no reference to the object exists.

Automatic garbage collection

- The object Q does not have a reference and cannot be used in future.
- The object becomes a candidate for automatic garbage collection.
- Java automatically collects garbage periodically and releases the memory used to be used in the future.



The this keyword

- The this keyword can be used inside any method to refer to the current object.
- Example:

```
Box (double w, double h, double d)
{
this.width = w;
this.height = h;
this.depth = d;
}
```

- Application-Instance Variable Hiding

```
Box (double width, double height, double depth)
{
this. width = width;
this. heigth = height;
this. depth = depth;
}
```


ALL YOUR DREAMS CAN
COME TRUE IF YOU HAVE THE
COURAGE
TO PURSUE THEM
Walt Disney