SWEN20003
Object Oriented Software Development

Classes and Objects 1

Bach Le
bach.le@unimelb.edu.au

University of Melbourne

The Road So Far

Lectures

- Subject Introduction
- A Quick Tour of Java

Learning Outcomes

Upon completion of this topic, which includes three lectures, you will be able to:

- Explain the difference between a *class* and an *object*
- Create classes, give them *properties* and *behaviours*, implement and use simple classes
- Identify a series of well-defined classes from a *specification*
- Understand the role of *getters*, *setters* and *constructors*
- Understand the differences between *instance*, *static* and *local* variables
- Understand the role of *standard methods* in java
- Explain object oriented concepts: *abstraction, encapsulation, information hiding and delegation*
- Understand the role of *wrapper* classes

Overview

This topic will be delivered through three lectures (Lectures 3, 4 and 5) each covering the following subtopics.

Classes and Objects - 1

- Introducing Classes and Objects
- Defining Classes
- Using Classes

Classes and Objects - 2

- Getters, Setters and Constructors
- Static Attributes and Methods
- Standard Methods in Java

Classes and Objects - 3

- Introducing Java Packages
- Information Hiding
- Delegation through Association
- Wrapper Classes

Introducing Classes and Objects

Introduction

All programming languages support four basic concepts:

- Calculation: constants, variables, operators, expressions
- Selection: `if-else`, `switch`, `?`
- Iteration: `while`, `do`, `for`
- Abstraction: The process of creating self-contained units of software that allows the solution to be parameterized and therefore more general purpose

Abstraction is the fundamental concept that differentiates procedural programming languages such as C from Object Oriented languages such as Java, C++.

Abstraction in Procedural Languages

Abstraction in procedural languages is provided through *functions* or *procedures*.

Functions manipulate external data by performing operations on them.

Example of a function in C that calculates the average of two floating point numbers:

```
1       float calculate_average (float a, float b) {
2           float result;
3           result = (a + b)/2;
4           return result;
5       }
```

Abstraction in Object Oriented Languages

Abstraction in Object Oriented (OO) languages is provided through an *Abstract Data Type (ADT)*, which contains *data* and *functions* that operate on data.

In Java a **Class** is an implementation on an **Abstract Data Type**.

Classes

- A "generalization" of a real world (or "problem world") entity
  - A physical real world thing, like a student or book
  - An abstract real world thing, like a university subject
  - An even more abstract thing like a list or a string (data)
- Represents a template for things that have common properties
- Contains *attributes* and *methods*
- Defines a new **data type**

Keyword

Class: Fundamental unit of abstraction in *Object Oriented Programming*. Represents an "entity" that is part of a problem.

Objects

- Are *instances* of a class
- Contain **state**, or dynamic information
- "**X** is of type A", "**X** is an object of the class A", and "**X** is an instance of the class A" are all equivalent

Keyword

Object: A specific, concrete example of a class

Keyword

Instance: An object that exists in your code

Motivating Example

Throughout this topic we will be referring to the following specification:

Develop a system (a set of classes) for a simple Drawing Pad application. The application should allow drawing different types of shapes, such as circles, squares, rectangles, display their geometrical properties: e.g. area, circumference. It should also allow different types of actions such as moving, resizing to the performed on shapes.

How would you develop this, right now? What additional information do you need?

Motivating Example

Throughout this topic we will be referring to the following specification:

Develop a system (a set of classes) for a simple Drawing Pad application. The application should allow drawing different types of shapes, such as circles, squares, rectangles, display their geometrical properties: e.g. area, circumference. It should also allow different types of actions such as moving, resizing to the performed on shapes.

How would you develop this, right now? What additional information do you need?

Drawing Pad Application - Classes

What classes can we use for our example problem?

Fundamental:

- `Drawing Pad`
- `Circle`
- `Square`
- Other shapes

Additional:

- `Drawing Tool`
- `Paint Brush`
- `Fill Colour`
- `Fill Type`
- Many more

Identifying Attributes and Methods

Let us consider the `Circle` class.

Can you identify the *attributes* and *methods* of the class?

Attributes:

- `Centre`
- `Radius`
- `Fill Colour`, `Fill Type`
- Many more

Methods (Operations):

- `Compute Circumference`
- `Compute Area`
- `Move`
- `Resize`
- Many more

Object Oriented Features

Following are some key features of the object oriented design paradigm:

- Data Abstraction
- Encapsulation
- Information Hiding
- Delegation
- Inheritance
- Polymorphism

Object Oriented Features

Following are some key features of the object oriented design paradigm:

- Data Abstraction
- Encapsulation
- Information Hiding
- Delegation
- Inheritance
- Polymorphism

Data Abstraction

Keyword

Data Abstraction: The technique of creating new data types that are well suited to an application by defining new classes.

A class is a special kind of programmer-defined data type.

- For example, by creating classes such as `Circle`, `Drawing Pad`, you are creating new data types, that can be used in applications.

A class is somewhat close to a structure in C but have additional features - attributes and methods.

The class definition determines the types of data (attributes) that an object can contain, as well as the actions (methods) it can perform.

Encapsulation

Keyword

Encapsulation: The ability to group data (attributes) and methods that manipulate the data to a single entity though defining a class.

A class encapsulates data and the methods that operate on the data into a single unit.

This method of encapsulation is unique to OO programming and is not provided by the procedural programming paradigm.

Defining a Class

Defining a Class

Syntax:

```
<visibility modfier> class <ClassName> {
    <attribute declarations>
    <method declarations>
}
```

A bare bone class:

```
// Circle.java - Circle class definition
public class Circle {

}
```

Defining a Class - Adding Attributes

Attribute Syntax:

```
<visibility modifier> <type> <variable name>;
```

Adding attributes (also called data, fields) to the `Circle` class.

```
// Circle.java - Circle class definition
public class Circle {
    public double centreX; //centre x coordinate
    public double centreY; //centre y coordinate
    public double radius;  //radius
}
```

Defining a Class - Adding Attributes

The attributes added to the `Circle` class in the above example are referred to as *instance variables*.

- these attributes maintain the state of the object; i.e. by giving values to `centreX, centreY, radius` we define a Circle object with particular size and position.

Keyword

Instance Variable: A **property** or **attribute** that is unique to each *instance (object)* of a class.

Defining a Class - Adding Methods

Method Syntax:

```
1       <visibility modifier> <void or typeReturned> myMethod(paramList)
2       {
3           variable declarations
4           statements
5       }
```

- If the method returns data, the data type must be specified in the method definition, otherwise, it is defined as `void`.
- If the method returns data, the method body must contain a return statement, which returns a variable of the specified return type.
- Variables can be declared inside the method - such variables are called *local variables*.
  Note: Local variables are inside the method as opposed to the instance variables (introduced earlier) which are outside the method declaration.

Defining a Class - Adding Methods

Adding methods to `Circle` class.

```
1       // Circle.java
2       public class Circle {
3           public double centreX;
4           public double centreY;
5           public double radius;
6
7           public double computeCircumference () {
8               double circum = 2 * Math.PI * radius;
9               return circum;
10          }
11          public double computeArea () {
12              double area = Math.PI * radius * radius;
13              return area;
14          }
15          public void resize (double factor) {
16              radius = radius * factor;
17          }
18      }
19
```

Using a Class

Using the `Circle` class

Follow the steps below to use the `Circle` class we just created.

- Create a file `Circle.java` and write the code.
  Note: the file name should match the class name.
- You can use an Integrated Development Environment (IDE), such a IntelliJ for this (will be introduced in the workshops), but in this instance use a text editor such as `notepad, wordpad, vim, kate etc.`
- Compile the class using the following command:

  ```
  javac Circle.java
  ```

  This creates a file `Circle.class`
- `Circle` becomes a derived data type that can be used in a Java program.

Using the Circle class

By creating the Circle class, you have created a new data type Circle - **Data Abstraction**.

- Variables of type Circle can be now defined in a program
- Circle is a **Derived Data Type** (as opposed to a Primitive Data Type such as int, float)

Example:

```
1  /* CircleTest.java: A test program to test the Circle class */
2  public class CircleTest {
3      public static void main(String args[]) {
4          Circle aCircle;
5          Circle bCircle
6      }
7  }
```

Using the `Circle` class

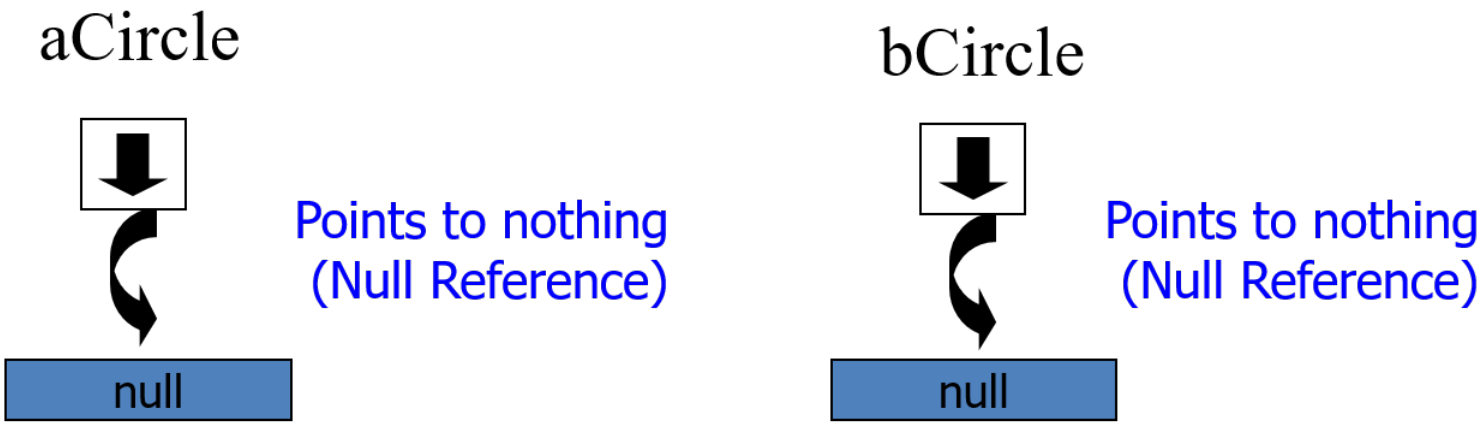
The declarations:

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

in the previous example did not create `Circle` objects.

`aCircle` and `bCircle` are simply **references** to `Circle` objects (not objects):

- Currently they point to nothing, hence **null references**.



The `null` Reference

Keyword

null: The Java keyword for "no object here". Null objects can't be "accessed" to get variables or methods, or used in any way.

Instantiating a Class

Objects are **null** until they are *instantiated*.

Keyword

Instantiate: To create an object of a class

```
1          // Instantiate an Circle object
2          Circle circle_1 = new Circle();
```

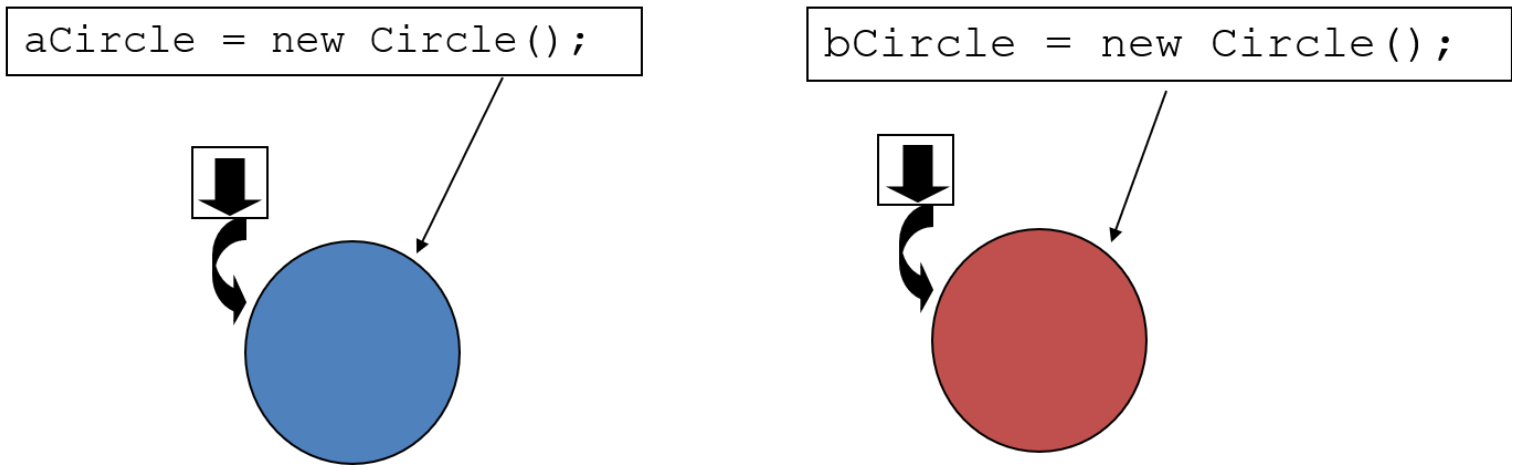
Keyword

new: Directs the JVM to allocate memory for an object, or *instantiate* it

Creating Objects

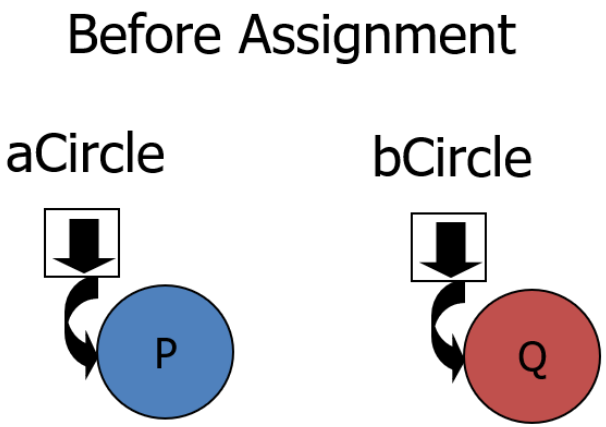
Objects are created dynamically using the `new` keyword.

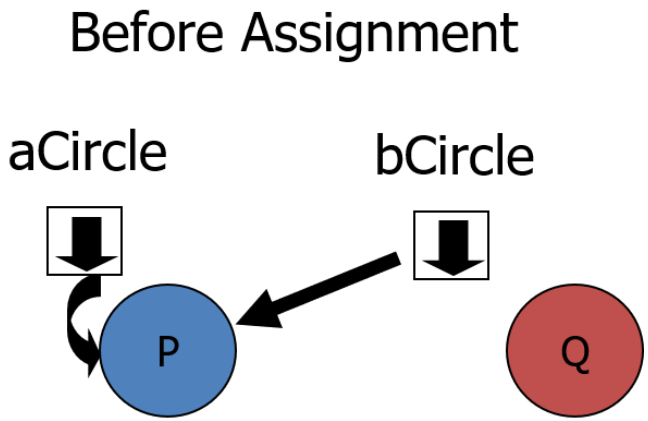
```
1  // CircleTest.java: A test program to test the Circle class
2  public class CircleTest {
3      public static void main(String args[]) {
4          Circle aCircle, bCircle;
5          aCircle = new Circle(); //aCircle now points to an object
6          bCircle = new Circle(); //bCircle now points to an object
7      }
8  }
```



Assigning References of a Class

```
1  // CircleTest.java: A test program to test the Circle class
2  public class CircleTest {
3      public static void main(String args[]) {
4          Circle aCircle, bCircle;
5          aCircle = new Circle(); //aCircle now points to an object
6          bCircle = new Circle(); //bCircle now points to an object
7          bCircle = aCircle; // Assining a class reference
8      }
9  }
```





Garbage Collection in Java

- In the previous example, object **Q** does not have a valid reference and, therefore, cannot be used in future.
- The object becomes a candidate for **Java Automatic Garbage Collection**.
  - Java automatically collects garbage periodically, and frees the memory of unused objects and makes this memory available for future use; you do not have to do this explicitly in the program.

Using Instance Variables and Methods

Syntax:

```
1       <objectName>.<varibaleName>;
2       <objectName>.<methodName>(<arguments>);
```

Syntax is similar to C syntax for accessing data defined in a structure.

Example:

```
1       Circle aCircle = new Circle();
2       double area;
3
4       // Initialize centre and radius
5       aCircle.centreX = 2.0;
6       aCircle.centreY = 2.0;
7       aCircle.radius = 1.0;
8
9       //Invoking methods or sending a "message" to methods
10      area = aCircle.computeArea();
11      aCircle.resize(2.0);
12
```

Using the Circle Class - Example

```java
// CircleTest.java - Test program to test the Circle class
public class CircleTest {
  public static void main(String args[]) {
    Circle aCircle = new Circle();
    aCircle.centreX = 10.0;
    aCircle.centreY = 20.0;
    aCircle.radius = 5.0;
    System.out.println("Radius = " + aCircle.radius);
    System.out.println("Circum: = " + aCircle.computeCircumference());
    System.out.println("Area = " + aCircle.computeArea());
    aCircle.resize(2.0);
    System.out.println("Radius = " + aCircle.radius);
  }
}
```

Program Output:

```
    Radius = 5.0
    Circum: = 31.41592653589793
    Area = 78.53981633974483
    Radius = 10.0
```

Back to the `main` method

- A program in Java is just a class that has a `main` method.
- When you give a command to run a Java program, the run-time system invokes the `main` method.
- The `main` is a `void` method, as indicated by its heading:

```
1          public static void main(String[] args) {
2
3          }
```

- `static` - it is still to come - please wait!

Learning Outcomes:

Topics covered in this lecture:

- Introducing Classes and Objects
- Defining Classes
- Using Classes

Learning Outcomes:

Upon completion of this lecture you will be able to:

- Explain the difference between a *class* and an *object*
- Create classes, give them *properties* and *behaviours*, implement and use simple classes
- Identify a series of well-defined classes from a *specification*
- Explain object oriented concepts: *abstraction and encapsulation*

References

- Absolute Java by Water Savitch (Fourth Edition), Chapters 4 & 5