CSC72003 Programming II

Topic 9

Author:
Dr Raina Mason, 2018

Topic 9
More about Classes

Introduction

This week we will be looking back at some of the concepts we have been using (without explanation so far) through this unit and the first programming unit. Understanding these concepts will help you to write your own classes, as you need to do in your major assignment.

In addition, some of these concepts are what makes object-oriented programming so powerful. In this topic and in next week's topic you will be moving forward into becoming more competent programmers.

Study materials

Materials required:

- The free BlueJ programming environment;
- Files in "Topic09" folder on MySCU site;
- Sample files from MySCU
- (Optional) Textbook sections 6.12 -> 6.16.

Objectives

On completion of this topic, you should be able to:

- use private and public methods and variables;
- explain the concept of information hiding;
- declare constants in Java
- declare static and instance variables in Java
- write static and object methods in Java
- create main methods
- create stand-alone Java applications

Concepts

- private
- public
- information hiding
- code completion
- static
- final
- constants
- main method
- java applications

Access modifiers

Access modifiers are the keywords **public** or **private** at the beginning of field declarations and method signatures.

For example:

```
private int numberOfSeats;


public void setAge(int replacementAge)
{ . . .
}


private int computeAverage()
{ . . .
}
```

Access modifiers define the visibility of a field, constructor or method. In the last topic, we looked at the interface for Java library classes, and their methods, constructors and fields. These were all **public** – which means that any other class can call these methods, use these constructors or use the fields.

If a field, constructor, or method is **private** then only the same class can use that field, constructor or method.

When we use a class, we need to know the interface, or public part of a class, so we have information on how to use it. We don't, however, need to know the implementation of the class – the private part of the class.

Information hiding

The internal implementation of a class is hidden from other classes. A programmer does not need to know the implementation in order to use the class. She only needs to know the interface.

Secondly, when a class is programmed, it should not be dependent on the internal implementation of another class. This defeats the purpose of modularisation. If the implementation of one class is changed, it should have no effect on other classes that use that class. This adheres to the good design principle of loose coupling. Using the private keyword means that we can prevent a class using the implementation details of another class.

Private methods and public fields

So far, most methods we have seen have been public methods. Sometimes, these could have been private methods. For example, in last week's topic, we implemented a TechSupport System. In the SupportSystem class, there were two methods **printWelcome()** and **printGoodbye()**. These methods were declared as private methods (see Figure next page):

```
/**
 * Print a welcome message to the screen.
 */
private void printWelcome()
{
    System.out.println("Welcome to the DodgySoft Technical Support System.");
    System.out.println();
    System.out.println("Please tell us about your problem.");
    System.out.println("We will assist you with any problem you might have.");
    System.out.println("Please type 'bye' to exit our system.");
}
```

```
/**
 * Print a good-bye message to the screen.
 */
private void printGoodbye()
{
    System.out.println("Nice talking to you. Bye...");
}
```

These methods will never be used from outside the class. They are simply used to break a larger task up into smaller ones. There is no reason for classes outside the SupportSystem class to "see" (be able to access) these methods. Therefore they are declared as private methods.

Private fields

In Java, fields can be declared as private or public. Generally, declaring fields as public breaks the information-hiding principles. It makes another class that is dependent on that information vulnerable to incorrect operation if the implementation changes.

Even though Java allows us to declare public fields, we consider this to be bad style and will always declare fields as private.

If access to a private field is channelled through accessor and mutator methods, then we can ensure that the field is never set to a value that would cause problems with our class. We can't do this if the field is made public.

Other access modifiers

There are two other access levels. One is the **protected** keyword, and one is when no access modifier is declared. These will not be covered in this unit.

The scribble demo

This project is provided so you can consolidate your learning so far. There are a number of exercises to do. Please don't skip over this section as it will help you to use the Library classes and provide a good example of the interface you should provide for your assignment.

Activity 9-1

Open and run the project scribble from the Chapter06 files. You run it by creating an object of class DrawDemo and experiment with its various methods.

The classes

The Canvas class provides a window on screen that can be used to draw on. It has operations for drawing lines, shapes and text. A canvas can be used by creating an instance interactively or from another object. You won't be modifying the Canvas class at all. It can be treated as a library class – open the editor and switch to the documentation view. This displays the class's interface with the Javadoc documentation.

The Pen class provides a pen object that can be used to produce drawings on the canvas by moving the pen across the screen. The pen itself is invisible but it will draw a line when moved on the canvas.

The DrawDemo class provides a few small examples of how to use a pen object to produce a drawing on the screen. The best starting point for understanding and experimenting with this project is the Draw Demo class.

Activity 9-2

Read the DrawDemo source code and describe in writing how each method works:

drawSquare() _____
_____
_____
_____.

drawWheel() _____
_____
_____
_____
_____

square() _____
_____
_____
_____
_____

colorScribble() _____
_____
_____
_____
_____

clear() _____
_____
_____
_____

Activity 9-3

Create a Pen object interactively using its default constructor (the constructor without parameters). Experiment with its methods. Keep the editor window open showing the documentation of the Pen class while you do this. Is the documentation effective?

Activity 9-4

Create a Canvas object interactively and try some of its methods. Again, refer to the class's documentation while you do this. Is the documentation effective?

Color

Some of the methods in the classes Pen and Canvas refer to parameters of type Color. This type is defined in class Color in the java.awt package. The Color class defines some colour constants, which we can refer to as follows:

```
Color.RED
```

Of course, if we want to use a library class, we need to import the Color class in the class that uses it.

Activity 9-5

Find some uses of the Color constants in the code of class DrawDemo. Write down four more colour constants that are available in the Color class. Refer to the class's documentation to find out what they are:

_____
_____
_____

Note that if we want to use the Color constants in the BlueJ method call dialog boxes, we have to fully qualify them with the class.

For example: if we were setting the pen colour using the setColor method, we would have to enter **java.awt.Color.RED** in the dialog box, not just "RED".

Activity 9-6

Create a canvas. Using the canva's methods interactively, draw a red circle near the centre of the canvas. Now draw a yellow rectangle.
How do you clear the whole canvas?

Activity 9-7

In class DrawDemo, create a new method called drawTriangle. This method should create a pen (as in the drawSquare method) and then draw a green triangle.

Activity 9-8

In class DrawDemo, write a method drawPentagon that draws a pentagon (a 5-sided figure).

Write a method drawPolygon(int n) that draws a regular polygon with n sides. So, n = 3 draws a triangle, n = 4 draws a square, etc.

Code completion

We have now been using library classes frequently. Sometimes we need to use a class's method but cannot remember the exact name or parameters. Most development environments offer *code completion* to help with this situation.

Code completion is available in BlueJ when the cursor is behind the dot of a method call. Pressing <CTRL>+<SPACE> will bring up a popup listing all methods in the interface of the object we are using in the call.

We can then type the beginning of the method name to narrow the listing. Hitting <RETURN> enters the selected method.

Activity 9-9

In class DrawDemo, type myCanvas.er and then press Ctrl-Space (with the cursor immediately behind the typed text) to activate code completion. How many methods are shown?

Add a method to your DrawDemo class that produces a picture on the canvas directly (without using a pen object). The picture can show anything you like, but should at least include some shapes, different colours, and text. Use code completion in the process of entering your code.

Class variables and constants

In the first programming unit, we looked at the concept of Java constants – which we declared with the two keywords **static** and **final**.

An example of a declaration of a constant is:

```
private static final int GRAVITY = 3;
```

You may recall that we always name constants in capitals, to indicate that their value cannot be changed.

You know what the access modified **private** means, but we will now examine the other two parts of this declaration: **static** and **final**.

Static keyword

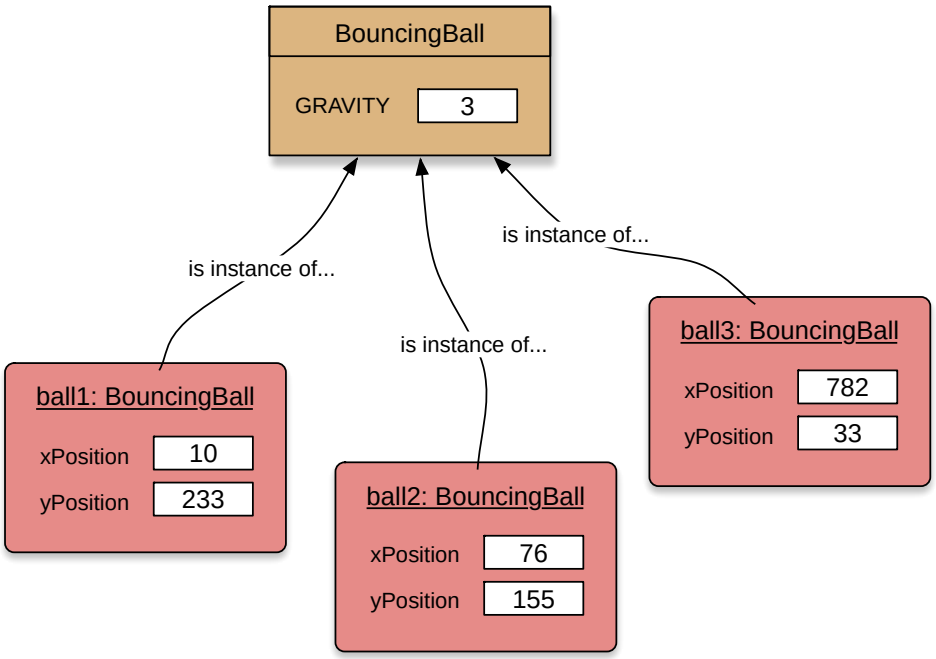
The static keyword is Java's syntax to define ***class variables***. Class variables are stored in the class itself, instead of one of the objects created from the class. This makes them quite different from

the instance variables (fields) that we have encountered so far.

Let's look at some example code:

```
public class BouncingBall
{
   private static final int GRAVITY = 3;
   private int xPosition;
   private int yPosition;
}
```

Now imagine we create three instances from this class.



As you can see from this diagram, the instance variables / field values of xPosition and yPosition are stored in each object. The class variable GRAVITY, is stored in the class itself. There is only ever one copy of this information, no matter how many objects are created.

The class variable can be accessed from any of the objects created from the class – in effect, the objects share this variable. We can use class variables when one value should always be the same for each object of a class.

Class methods

Java also supports class methods (also known as static methods) which are methods that belong to a class. We will come back to these after discussing the **final** keyword.

Constants

We often use static in conjunction with the final keyword, but final can also be used on it's own. For example:

```
private final int SIZE = 10;
```

This defines a constant named SIZE with the value 10. You can see that the only differences between a constant declaration and a normal field declaration is that:

- the keyword final is included before the type name; and
- the constant must be initialised with a value when it's declared.

Its good practice to declare any fields that we do not ever want to change as constants. This makes sure it cannot be accidentally changed later.

Constants usually apply to all instances of a class, so we usually declare them as class constants, with a static keyword. This ensures the constant is just stored in one spot.

```
private static final int SIZE = 10;
```

If you have a look in your scribble project, you will see two constants used in the Pen class. We also used constants from the Color class in that project, such as Color.RED. Note that the constants in the Color class were declared public, which allowed us to use them!

Activity 9-10

Write constant declarations for the following:

1. A public variable that is used to measure tolerance, with the value 0.001;

   _____

2. A private variable that is used to indicate a pass mark, with the integer value of 40:

   _____

3. A public character variable that is used to indicate that the help command is 'h';

   _____

Activity 9-11

What constant names are defined in the java.lang.Math class?

_____
_____
_____
_____

Activity 9-12

In a program that uses the constant value 73.28166 in ten different plaes, give reasons why it makes senses to associate this value with a variable name.

_____
_____
_____
_____

Class methods

All of the methods we have seen so far have been instance methods – that is, they are invoked on an instance of a class.

A class method can be invoked without an instance (an object) – just having the class is enough.

Just like class variables, class methods are defined by adding the keyword static in front of the type name in the method's header:

```
public static int getNumberOfDaysThisMonth()
{
   // code here
}
```

To call a class method, we use the class name before the dot in the usual dot notation.

For example, if the method above was defined in a class called Calendar, then we would call it like this:

```
int days = Calendar.getNumberOfDaysThisMonth();
```

Note that no Calendar object was created.

Activity 9-13

Read the class documentation for class Math in the package java.lang. It contains many static methods (class methods). Find the method that computes the maximum of two integer numbers. What does its header look like?

_____

Why do you think the methods in the Math class are static? Could they be written as instance methods?

Activity 9-14

Write a class that has a method to test how long it takes to count from 1 to 10000 in a loop. You can use the method currentTimeMillis from class System to help with the time measurement.

are shown?

Activity 9-15

The Collections class in the java.util package contains a large number of static methods that can be used with collections such as ArrayList, LinkedList, HashMap and HashSet. Read the class documentational for the min, max and sort methods, for instance. While you might not fully understand the explanations or the notation used, being aware that these methods exist will be

useful to you.

Limitations of class methods

Because class methods are associated with a class rather than an instance, they have two important limitations.

The first is that a class method may not access any instance fields defined in the class. This makes sense, because instance fields are associated with individual objects. Class methods are restricted to accessing class variables from their class.

The second is that a class method may not call an instance method from the class. The class method may only invoke other class methods defined in the class.

Executing without BlueJ

As we have stated previously, to execute an application without BlueJ, we need a class with a Main method. If we look closer at this method's header you will now understand more of the keywords:

```
public static void main(String [] args)
```

The **public** means that this method can be accessed from outside the class – important if this is the method that is automatically called when a program is executed.

The **static** means that this is a method that belongs to the class itself – we don't need to instantiate any objects of the class to call the method.

The **void** keyword means that nothing is returned from this method

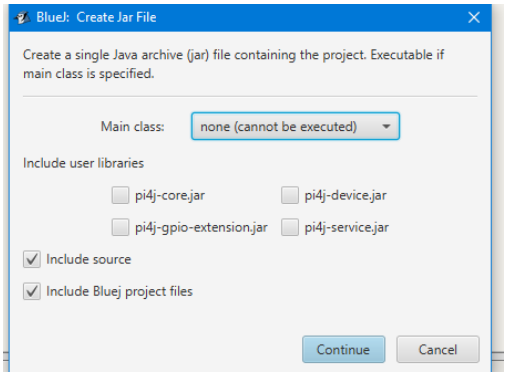
The **main** name/keyword is a special method name that says this is the method that you need to call first in a program, once the beginning class is identified.

The **String [] args** names an array called args of Strings – which is one way we can feed values into this program. We won't look any further at this way of entering values in this unit.

Creating an application

We can create an application in BlueJ by choosing "Create Jar file" from the Project menu.

A dialog box will open, asking the programmer to specify the class with the main method:



If we have created a main method in our first class, then we can specify this class, and follow the prompts to create a Java executable.

Activity 9-16

Add a main method to your SupportSystem class in the tech-support project. The method should create a SupportSystem object and invoke the start method on it. Test the main method by invoking it from BlueJ. Note that class methods can be invoked from the class's popup menu.

Execute the program without BlueJ.

Can a class count how many instances have been created of that class? What is needed to do this? Write some code fragments that illustrate what needs to be done. Assume that you want a static method called numberOfInstances that returns the number of instances created.

Workshop activities

Your workshop activity this week is a choice of three activities:

1) continue work on your assignment;
2) complete the below activities;
3) documenting your assignment code to be compatible with Javadoc;
4) continue on activities you have not completed in the topic.

Activity R9-1

Open the bouncing-balls project and find out what it does. Create a BallDemo object and execute the bounce() method.

Change the method bounce in class BallDemo to let the user choose how many balls should be bouncing.

Activity R9-2

For this exercise, you should use a collection to store the balls. This way, the method can deal with 1, 3 or 75 balls - any number you want. The balls should initially be placed in a row along the top of the canvas.

Which type of collection should you choose? So far, we have seen an ArrayList, a HashMap and a HashSet.

Activity R9-3

Change the bounce() method to place the balls randomly anywhere in the top half of the screen.

Activity R9-4

Write a new method named boxBounce. This method draws a rectangle (the box) on screen and one or more balls inside the box. For the balls, do not use BouncingBall, but create a new class BoxBall that moves around inside the box, bouncing off the walls of the box so that the ball always stays inside. The initial position and speed of the ball should be random. The boxBounce method should have a parameter that specifies how many balls are in the box.

Activity R9-5

Give the balls in boxBounce random colours.