SWEN20003
Object Oriented Software Development

Classes and Objects 3 - Questions

Bach Le
bach.le@unimelb.edu.au

University of Melbourne
© University of Melbourne 2023

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

Answer true or false:

Information hiding allows restricting methods in a class accessing other attributes and methods.

Answer:
False - it only allows control external access to the class.

Name two visibility modifiers for attributes and methods and explain their role?

Answer:
private - only visible within the class
public - visible to anybody - inside the class or anybody using the class

What happens if we do not specify a visibility modifier?

Answer:

Define the term "Information Hiding".

Answer:
Ability to hide attributes and methods from the person using a class in their code.

Why is "Information Hiding" needed?

Answer:
Information hiding allows us to fully control what can and can't be accessed/modified by others outside of our class.

Which of the following statements related to delegation/association are correct?

1. Through delegation, the functionality of a class can be extended.
2. To delegate the class must have access to a reference of the class delegated to.
3. When a class contains another class as an attribute, an association relationship between the two classes is formed.
4. Delegation promotes code reuse.

Answer:
1, 2, 3, 4

What is the difference between implicit and explicit typecasting?

Answer:

Implicit typecasting occurs when a type can be safely converted into another, like `int` to a `double`.

```
int a = 5;
double b = a;
```

Explicit typecasting is used when the conversion is not considered a safe operation and may result in the loss of data. You have to explicitly tell Java to perform the type conversion:

```
double a = 3.1415;
int b = (int)a; // b = 3, we lose our fractional information
```

What is boxing and unboxing?

Answer:
Boxing occurs when a primitive value is converted into a wrapper object, e.g. int to Integer.

```
int a = 3;
Integer b = a;
```

Unboxing occurs when a wrapper object is converted into its primitive type, e.g. Integer to int.

```
Integer a = new Integer(3);
int b = a;
```

Is boxing/unboxing done implicitly or explicitly?

Answer:
Boxing and unboxing are both done implicitly where possible. Explicit boxing only needs to be done when you want to use a method of the wrapper class on a primitive value. Here is an example of explicit boxing:

```
int a = 1;
String stringRepresentation = ((Integer)a).toString();
```

A better way would be simply String.valueOf(a)

Assess Yourself

Design a class, including attributes, method names, and constructors, for a *drinking glass*.

What things describe/define these objects?

What can an object of this class do? What can *other objects* do *with/to* this object?

Assess Yourself

Firstly, we add the attributes:

```
public class Glass {
    // Easy stuff; primitive attributes
    private double height, radius;

    // More interesting; initialising attributes
    private double fillHeight = 0;

    private boolean isFull = false;

    // Even more interesting; using
    // other classes as attributes
    private Material material;

    private Shape shape;
}
```

Assess Yourself

Add getters and setters:

```
public class Glass {
    public double getHeight() { return height;}
    public void setHeight(double height) { this.height = height;}
    public double getRadius() { return radius;}
    public void setRadius(double radius) { this.radius = radius; }
    public double getFillHeight() { return fillHeight; }
    public void setFillHeight(double fillHeight) {
                    this.fillHeight = fillHeight; }
    public boolean isFull() { return isFull; }
    public void setFull(boolean full) { isFull = full;}
    public Material getMaterial() { return material; }
    public void setMaterial(Material material) {
                    this.material = material; }
    public Shape getShape() { return shape;}
    public void setShape(Shape shape) { this.shape = shape; }
}
```

Assess Yourself

Next, we use the attributes to add a constructor:

```
public class Glass {
    // Note how we don't have to include the
    // variables we already initialised
    public Glass(double height, double radius,
        Material material, Shape shape) {
        this.height = height;
        this.radius = radius;
        this.material = material;
        this.shape = shape;
    }
}
```

Assess Yourself

We can also have multiple constructors:

```
public class Glass {
    public Glass(double height, double radius,
        Material material, Shape shape, double fillHeight) {
        this.height = height;
        this.radius = radius;
        this.material = material;
        this.shape = shape;
        this.fillHeight = fillHeight;

        // Note: approxEqual needs to be written by us
        if (approxEqual(this.height, this.fillHeight)) {
            this.isFull = true;
        }
    }
}
```

Assess Yourself

Finally, we add methods so our objects can do things:

```
public class Glass {
    public void fillGlass() {
        fillHeight = height;
        isFull = true;
    }

    public void emptyGlass() {
        fillHeight = 0;
        isFull = false;
    }

    public double calculateVolume() {
        return this.shape.calculateVolume();
    }
}
```

Assess Yourself

Finally, we add methods so our objects can do things:

```
public class Glass {
    // Note: it would be more appropriate to define
    // this in a "utility" class
    public static boolean approxEqual(double var1,
        double var2) {
        return Math.abs(var1 - var2) < 0.00001;
    }

    public static String message() {
        return "I am a Glass";
    }

    // Why are these both static?
}
```

Assess Yourself

Design classes, including attributes and methods, for the following scenario:

An up and coming entrepreneur wants your advice on their latest venture: a system that allows "decision makers" like local governments to import, view, and visualise data.

The system should be able to read CSV and XLS documents, should be able to present the data in a table with appropriate filters, and also visualise the data with graphs, charts, etc.

Assess Yourself

Class: `Data`

- Attributes
  - values
  - nRows
  - nCols
- Methods
  - readData
  - add
  - remove

Assess Yourself

Class: `Display`

- Attributes
  - values
  - nRows
  - nCols
- Methods
  - filter
  - delete

Assess Yourself

Class: `Chart`

- Attributes
  - values
  - width
  - height
  - colours
- Methods
  - build
  - update
  - changeColours

Assess Yourself

Design classes, including attributes and methods, for the following scenario:

As a software engineer at RobotOverlords Inc., you've been tasked with designing the software systems for Murder Bot V3, your flagship fly-swatting robot.

Murder Bot has a number of sensors (battery, GPS, motors), actuators (arms, legs), and controls (electric swatter, flamethrower).

Assess Yourself

Class: `Sensor`

- Attributes
  - value
  - name
- Methods
  - measure
  - calibrate

Assess Yourself

Class: `Actuator`

- Attributes
  - position
- Methods
  - calibrate
  - measurePosition

Assess Yourself

Class: `Control`

- Attributes
  - isOn
  - batteryLevel
- Methods
  - activate
  - measureBattery

Assess Yourself

Design classes, including attributes and methods, for the following scenario:

You have been asked to develop the user interface (and associated backend) for a shopping centre's in-store map system.

The system should allows users to search for stores, find directions to stores, and list stores and their details.

Assess Yourself

Class: `Interface/UserInterface`

- Attributes
  - shops
  - map
- Methods
  - search
  - getDirections
  - listStores

Assess Yourself

Class: `Shop`

- Attributes
  - type
  - contactName
  - contactNumber
  - location
- Methods
  - openingHours

Assess Yourself

Class: `Map`

- Attributes
  - level
  - shops
  - source
  - destination
- Methods
  - changeLevel
  - displayRoute
  - overlayText

Assess Yourself

Class: `CustomerInterface`

- Attributes
  - shops
  - map
  - directory
- Methods
  - search
  - getDirections
  - listStores

Assess Yourself

A world class chef is working with you to develop a robotic assistant, and they've suggested you build a simulated kitchen to test it out.

The robot needs to be able to:

- Use normal human tools and utensils (oven, fork)
- Open things, like cupboards and containers
- Prepare ingredients in various ways
- Be operated by a human (for safety reasons)

How would you develop this system? What classes would you use? What methods and attributes would they have? What interface does your program have between the user and the robot?

References

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

The Road So Far

Lectures

- Subject Introduction
- A Quick Tour of Java
- Classes and Objects - 1, 2

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