SWEN20003
Object Oriented Software Development

Classes and Objects 2 - Questions

Bach Le
bach.le@unimelb.edu.au

University of Melbourne

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

What are getters and setters in Java? Why are they needed?

Answer:

- They *encapsulate* variables in our classes by exposing read and write functionality through methods rather than direct access.
- Encapsulation allows us to easily make future changes to our get/set logic of a variable. For example, say we have a setter `setAuthorizationLevel(String level)`. This setter controls the authorisation level of a user in some secure system. Instead of just changing a variable, the setter might perform the following actions:
  - Check if the authorisation level exists.
  - If the authorization level is particularly high, perform multi-factor authentication
  - Change the authorization level
  - Email/text user that their authorization level has been changed
- To ensure proper encapsulation, the variable must be only available for direct modification from its enclosing class through the inclusion of the `private` privacy modifier - next lecture.

What is the purpose of a constructor, and how do you use one?

Answer:

- A constructor *constructs* (or builds) an **object**, by initialising a new instance of the specified class.
- A constructor that takes parameters is known as a parameterized constructor: parameterized constructors enable the instantiator to provide initial values to the object.
- If a class does not specify a constructor, Java provides a "default constructor" that has no parameters.
- We use a constructor when creating an object using the `new` keyword.

Which of the following statements regarding a static attribute are true:

1. Does not belong to any specific instance of a class
2. Cannot be modified using a reference to an object
3. Memory is allocated at compile time
4. Shared across objects of the of the class
5. Can be accessed and modified by an instance (non-static) method
6. Can be accessed and modified by an static method
7. Cannot be returned by a instance (non-static) method

Answer:

1,3, 4, 5, 6

Which of the following statements regarding a static method are true?

1. Can be accessed using a reference to an object
2. Can access and modify instance variables
3. Can access and modify static variables
4. Can return static variables
5. Can return instance variables
6. Can call static methods within the class
7. Can call instance methods within the class

Answer:

1, 3, 4, 6

When designing a class how would you decide whether a particular method should be static or non-static?

Answer:
Ask yourself the following question.
Does the method depend on any instance variables of the class - if not make it static?
This does not mean that you remove instance variables from the class and make them parameters to the method to make the method static - that is going back to procedural programming.

Does is make sense to have a class with no instance variables, and all methods defined static?

Answer:
Yes - to group a set of related methods.
Example:
Math class.

What does the keyword *this* mean?

Answer:

The keyword `this` refers to the *calling object* itself.

State typical uses of the keyword `this`:

Answer:

1. To refer to an instance variable of the class
2. For a constructor to call another constructor within the class
3. To pass a reference to the object itself

What is the purpose of the `toString()` method?

Answer:
To provide a meaningful description of the object when it is output using the reference to the object - when the object reference is used in a place a `String` is expected, the `toString()` method gets called automatically.

What happens if you use the object reference in a place a `String` is expected, but the class does not have a `toString` method?

Answer:
Class name followed by a hashcode that uniquely identifies the object gets printed e.g. Circle@1540e19d