Lecture 1: Object-Oriented Design and Methodology/Object Oriented Concepts

1 Introduction

1.1 Course Details

- ICCS330 Object-Oriented Design and Methodology
- ICCS428 Object Oriented Concepts
- Instructor: Weerapong Phadungsukanan (weerapong.pha@mahidol.edu)
- Office Hours: Posted office hours or by appointment in 1409
- We use Canvas for gradebook, homework submissions, discussions, etc. Please enroll at https://canvas.instructure.com/enroll/TT4JPL.

1.2 Overviews

Software engineer is one of the most demanding area in today business. Thousands of techniques, design patterns, concepts, libraries and frameworks have been well established and have been extensively used over the last few decades. In this course, you will be learning some of these useful important topics for developing a medium-scale multi-tier application. After completing this course, you **MUST**:

- Understand the fundamental software development life cycle.
- Understand the concept of object-oriented programming and design patterns.
- Be able to use UML diagram to describe their object-oriented system.
- Be able to read and write application or third-party library documentations.
- Have experience designing and developing a medium-scale multi-tier application.
- Have experience testing, debugging and analyzing your software.
- Have experience in an iterative and incremental software development methods.

1.3 Textbook

For design concepts and methodology:

- System Analysis and Design, An Object Oriented Approach with UML, Alan Dennis, Barbara Haley Wixom, David Tegarden.

For object-oriented programming, Java, and others, we are not following any particular textbook. All the key topics should already be in the lecture notes and, therefore, you should be able to find the references you need on the Internet. Keep in mind that many of these technologies are rapidly changing so it is recommended for you to subscribe to websites or online publishers for news and updates, e.g. DZone, InfoQ, etc.

1.4 General Expectations

- You are well-versed in Java and Object Oriented Programming.
- Work hard, by the end, you will have learned a lot.
- Do your best, especially for the projects. It will be a useful portfolio for you.
- Ask questions to help you learn.

1.5 Grading scheme

Score distribution is given by:

25% Assignments.

40% Project. You will be working in a small group to develop an application of your choices. However, the project proposal must be submitted first to get an approval. Details will be given later. This is supposed to be fun.

15% Midterm exam.

15% Final exam.

5% Participation, etc.

For two exams:

- They will be paper based examinations.
- No books, notes, or consultation with any other person allowed.

1.6 Assignments and Late Policy

- There will be assignments given on regular basis.
- Each assignment **MUST** be submitted to Canvas on the specified date/time.
- Encouraged to hand them in well ahead of the deadline.
- You are allotted **FOUR (4)** late days for the term at no grade penalty.
- At most **ONE (1)** late day may be used per assignment at no grade penalty.
- Whichever assignment that uses 2, 3 and 4 late days will be multiplied by 0.8, 0.6 and 0.4 respectively.
- Late days are applied automatically, you cannot choose which assignment to use the late days for. This is to prevent you from optimizing for the best score.
- This policy is not negotiable, except for legitimate reason, e.g. serious illness that requires hospitalized or impede the ability to work.

1.7 Laptop vs Lab Computers

The lab environment is here for you to use. We also recommend that you bring and use your laptop. It is simply easier to take notes and work on assignments using a machine that you have full control over.

- We are using Java 8. Download and install the latest Java 8 SDK (specifically, Java SE Development Kit 8).
- A good IDE is essential in software development so you can pick any tools that are best for you. IntelliJ IDEA will be used for teaching in this course, but you can choose a different one.

2 Short course on Java

2.1 Java

Here is a simple Hello World application:

```java
package io.muic.ooc;
/**
 * This is the main class.
 * @author gigadot
 */
public class Hello {

    public static void main(String[] args) {
        System.out.println("Hello Worl");
    }
}
```

How to compile? Just type:

```
javac Hello.java
```

Notice a Hello.class file in the same folder. A class file is a binary or compiled code of the Java source code. To run this binary, we need Java Runtime Environment (JRE):

```
java Hello
```

Q: What is JDK?
A:
Q: What is the difference between JDK and JRE?
Q:

Imagine if you have hundred of Java files, what would you do? This is where a *build tool* come in. A few of well-known Java Build tools are Ant, Maven and Gradle, which are listed by their ages from the oldest to the newest. Some useful read up can be found at https://technologyconversations.com/2014/06/18/build-tools/.

2.2 Maven

In this course, we will be using Maven as our main build tools. Despite the fact that Gradle has started to become increasingly more popular, majority of third-party libraries and projects are still based on Maven. (But the main reason for choosing this is because the instructor is more familiar with the tools and can provide better supports to students.)

Maven uses conventions for the build procedures. An XML file is used to describe the dependencies on other external modules, components, and libraries. A set of well-defined build sequences are pre-defined in Maven plugins, which covers most of the common use cases. However, it is very difficult with Maven for any uncommon tasks and this is where Gradle deals with the problems nicely.

One of the best designs of Maven is that it hosts the third-party libraries online in the Maven 2 Central Repository and Maven automatically downloads and stores them in a local cache. Therefore, there is no need to store the third-party libraries locally and it is easy to update the dependencies.

Q: Where exactly does Maven store the local cache?
A:

Maven relies heavily on conventions, such as:

- Naming conventions of your project groupId, artifactId and version.
- Directory structure.
- Build life-cycle and goals.

Standard project structure

directory	description
/app/pom.xml	maven2 project file
/app/src/	Sources
/app/src/main/java/	Java source tree
/app/src/test/java/	Java unit tests
/app/src/main/resources/	Java classpath resources
/app/src/test/resources/	Resources for unit-tests
/app/target/	compiles classes and target files
/webapp/src/main/webapp	root of webapp

As you can see, creating this directory structure is boring so Maven provides you with command line tools, known as project *archetype*. You can use the standard archetype or even create your own ones. Here we will start with existing archetype. To generate a Jar project, type in terminal:

```
mvn archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes -DarchetypeArtifactI
```

Q: What is a Jar file?
A:
Q: How to build/compile the project?
A:
Q: How to clean the project folder?
A:
Q: What is the different between an application and a library?
A:

Note: You can creating a project and all these with IDE.

2.3 Naming conventions and style guild line and why are they important.

Naming is one of the hardest tasks and the best practice in programming.

2.3.1 Package names

All lower cases, separated by dot.

Acceptable:

- `package io.muic.ooc`
- `package io.muic.ooc.course`
- `package io.muic.ooc.user.service`

Unacceptable:

- `package io.muic.ooc_course`
- `package io.muic.ooc.user_service`

2.3.2 Class and Interface names

Use camel case, first letter is uppercase.

Acceptable:

- `class CourseService`
- `class OocMindblower`
- `interface OptionalBehaviour`

Unacceptable:

- `class Course_Service`
- `class ooc_mindblower`

2.3.3 Attribute, field, and local variable names

Use camel case, first letter is lowercase.

Acceptable:

- `int age;`
- `boolean hasDefaultValue;`
- `String responseText;`

Unacceptable:

- `int Age;`
- `boolean has_default_value;`

2.3.4 Constant names

All letters are uppercase, separated by underscore (_).

Acceptable:

- `final int MAX_INTEGER = 100;`
- `final String DEFAULT_STATUS = "done";`

Unacceptable:

- `final int maxInteger = 100;`
- `final String DEFAULTSTATUS = "done";`

2.3.5 Method names

Use camel case, first letter is lowercase.

Acceptable:

- `int getAge()`
- `Item findItemBySearchTerm(String searchTerm)`
- `void updatePositionX(int x)`

Unacceptable:

- `Item find_item_by_search_term(String search_term)`
- `void updatepositionx(int x)`

2.3.6 Getter and Setter

Class attributes should always be accessed through accessors and mutators (getters and setters). Accessors and mutators are methods used to return and set the attribute value. This is to provide encapsulation to the object. These methods typically begin with "get" or "set", followed by the attribute name. As with other methods, the first letter of the method name should be in lowercase.

Note: Boolean accessors typically begin with "is" i.e. `isFixed()`, and return a Boolean value.

Note: Getters take no arguments.

Example:

```
public class PopulationCounter {
    // Attribute:
    private int count;

    // Getter
    int getCount(){
        return count;
    }

    // Setter
    void setCount(int count){
        this.count = count;
    }

    void incrementCount() {
        setCount(getCount() + 1);
    }

    void decrementCount() {
        setCount(getCount() - 1);
    }
}
```

2.3.7 Returning Arrays and Lists

User plural in conjunction with the method naming convention.

Acceptable:

- `List<Person> getPeople()`
- `List<Item> getItems()`

Unacceptable:

- `List<Person> getPersons()`

2.4 OOP

Source: directly taken from
http://study.com/academy/lesson/oop-object-oriented-programming-objects-classes-interfaces.html

2.4.1 What Is an Object in Programming?

Object-oriented programming, or OOP, is an approach to problem solving where all computations are carried out using objects. An object is a component of a program that knows how to perform certain actions and how to interact with other elements of the program. Objects are the basic units of object-oriented programming. A simple example of an object would be a person. Logically, you would expect a person to have a name. This would be considered a property of the person. You could also expect a person to be able to do something, such as walking or driving. This would be considered a method of the person.

Code in object-oriented programming is organized around objects. Once you have your objects, they can interact with each other to make something happen. Let's say you want to have a program where a person gets into a car and drives it from A to B. You would start by describing the objects, such as a person and car. That includes methods: a person knows how to drive a car, and a car knows what it is like to be driven. Once you have your objects, you bring them together so the person can get into the car and drive.

2.4.2 Classes and Objects

A class is a blueprint of an object. You can think of a class as a concept, and the object is the embodiment of that concept. You need to have a class before you can create an object. So, let's say you want to use a person in your program. You want to be able to describe the person and have the person do something. A class called 'person' would provide a blueprint for what a person looks like and what a person can do. To actually use a person in your program, you need to create an object. You use the person class to create an object of the type 'person.' Now you can describe this person and have it do something.

Classes are very useful in programming. Consider the example of where you don't want to use just one person but 100 people. Rather than describing each one in detail from scratch, you can use the same person class to create 100 objects of the type 'person.' You still have to give each one a name and other properties, but the basic structure of what a person looks like is the same.

2.4.3 Methods and Functions

Once you have created objects, you want them to be able to do something. This is where methods come in. A method in object-oriented programming is a procedure associated with a class. A method defines the behavior of the objects that are created from the class. Another way to say this is that a method is an action that an object is able to perform. The association between method and class is called binding. Consider the example of an object of the type 'person,' created using the person class. Methods associated with this class could consist of things like walking and driving. Methods are sometimes confused with functions, but they are distinct.

A function is a combination of instructions that are combined to achieve some result. A function typically requires some input (called arguments) and returns some results. For example, consider the example of driving a car. To determine the mileage, you need to perform a calculation using the distance driven and the amount of fuel used. You could write a function to do this calculation. The arguments going into the function would be distance and fuel consumption, and the result would be mileage. Anytime you want to determine the mileage, you simply call the function to perform the calculation.

How does this differ from a method? A function is independent and not associated with a class. You can use this function anywhere in your code, and you don't need to have an object to use it.

Now, what if you were to associate the function with an object of the type 'car?' For example, you want to be able display the mileage of the car on the dashboard. In this case, the mileage calculation has become a method because it is a procedure associated with the car's class. Every time you create a new object of the type 'car' using the car class, this method will be part of the object. The action the car is now able to perform is to calculate mileage. It is the same calculation as performed by the stand-alone function but is now bound to the car.

2.4.4 OOP Concepts

Object-oriented programming is built around a number of concepts. These concepts are implemented using classes, objects and methods, but it is useful to review those concepts more generally. Four core concepts of object-oriented programming are abstraction, encapsulation, inheritance and polymorphism.

Using abstraction, a programmer hides many of the details about an object and shows only the most relevant information. This reduces complexity and increases efficiency. For example, in the case of a person, there could be any number of detailed descriptions. However, if only the name and age are really relevant in a particular context, only those descriptions will be used.