Programming for Mobile Computing

EECS 1022

`moodle.yorku.ca`

Midterm

Written part: Monday July 10 during the lecture (1 hour)

Programming part: Tuesday July 11 during the lab (1.5 hour)

Today's Lecture

- Anatomy of a Java class
- API
- JUnit

Anatomy of a Java Class

```
package ca.roumani.bmi;

/**
 * Model for the BMI app.
 *
 * @author Hamzeh Roumani
 */
public class BMIModel
{
    private double weight;
    private double height;

    ...
}
```

Package Statement

```
package ca.roumani.bmi;

/**
 * Model for the BMI app.
 *
 * @author Hamzeh Roumani
 */
...
```

The package statement consists of the keyword package followed by the name of the package. The name of a package consists of one of more sequences of letters separated by dots. A package groups a number of classes.

The name of the package is also reflected in the folder structure. For example, the folder ca contains a folder roumani which contains a folder bmi which contains all the classes belonging to the package ca.roumani.bmi.

Documentation Comments

```
package ca.roumani.bmi;

/**
 * Model for the BMI app.
 *
 * @author Hamzeh Roumani
 */
public  class  BMIModel
 ...
```

Each documentation comment starts with /** and ends with */. Documentation comments may contains tags such as @author.

The above documentation comment describes the class and mentions the author of the class.

Class Header

```
package ca.roumani.bmi;

/**
 * Model for the BMI app.
 *
 * @author Hamzeh Roumani
 */
public class BMIModel
 ...
```

The above class header consists of the keyword public followed by the keyword class followed by the name of the class.

A public class can be used by other classes. All the classes that we develop in this course will be public.

Class Body

```
...
public class BMIModel
{
  private double weight;
  ...
}
```

The separators { and } define a scope. Every attribute and method defined inside the { and } is not visible outside them.

Attribute Declarations

```
...
public class BMIModel
{
   private double weight;
   private double height;
    ...
}
```

The above attribute declarations consist of the keyword private followed by the type double followed by the name of the attribute.

Private attributes are only visible inside the body of the class. All the attributes that we declare in this course will be private.

Constructor Header

```
...
public class BMIModel
{
   ...
   public BMIModel(String weightText, String heightText)
   {
      ...
   }
}
```

The above constructor starts with the keyword public followed by the class name followed by the parameter list. A constructor can have zero or more parameters. Each parameter consists of a type and a name. The parameters are separated by commas.

Constructor Body

```
...
public class BMIModel
{
   ...
   public BMIModel(String weightText, String heightText)}
   {
      ...
   }
}
```

The separators { and } define a scope. Every variable declared inside the { and } is not visible outside them.

Documentation Comments

```
...
public class BMIModel
{
    ...
    /**
     * Returns the index of this model, with one digit precision.
     *
     * @return the index of this model.
     */
    public String getBMI()
    ...
}
```

The above documentation comment describes the method getBMI.
It uses the @return tag to describe what the method returns.

Method Header

```
...
public class BMIModel
{
    ...
    /**
     * Returns the index of this model, with one digit precision.
     *
     * @return the index of this model.
     */
    public String getBMI()
    ...
}
```

The above method header starts with the keyword public followed by the return type (that is, the type of the value returned by the method) followed by the parameter list. This method has zero parameters.

Method Body

```
...
public class BMIModel
{
   ...
   public String getBMI()
   {
      ...
   }
}
```

The separators { and } define a scope. Every variable declared inside the { and } is not visible outside them.

Return Statement

```
...
public class BMIModel
{
    ...
    public String getBMI()
    {
        double index = this.weight / (this.height * this.height);
        String indexText = String.format("%.1f", index);
        return indexText;
    }
}
```

The return statement consists of the keyword return followed by an expression. The type of this expression must match the return type of the method.

Variable Declarations

```
public class PingPongActivity
{
  public void buttonClicked(View view)
  {
    String text;
     ...
  }
}
```

The above variable declaration consist of the type of the variable followed by its name.

Assignments

```
public class PingPongActivity
{
  public void buttonClicked(View view)
  {
    String text;
    text = "Test";
     ...
  }
}
```

The above assignment consists of a variable name, the separator = and the value "Test". The type of the variable and the type of the value should be compatible.

Combining Declaration and Assignment

```
public class PingPongActivity
{
  public void buttonClicked(View view)
  {
    String text = "Test";
     ...
  }
}
```

Components of a Class

Attributes

The attributes capture the data.

Components of a Class

Attributes

The attributes capture the data.

Constructors

The constructors initialize the data.

Components of a Class

Attributes

The attributes capture the data.

Constructors

The constructors initialize the data.

Methods

The methods manipulate of the data.

Components of PingPongModel

Attributes

An attribute of type boolean.

Components of PingPongModel

Attributes

An attribute of type boolean.

Constructors

The constructor initializes the attribute to true.

Components of PingPongModel

Attributes

An attribute of type boolean.

Constructors

The constructor initializes the attribute to true.

Methods

The method getPingPong returns Ping, Pong, Ping, Pong, ...

API

The Application Programming Interface (API for short) provides a specification of the public features (attributes, constructors and methods) of a class. The API is generated from the documentation comments and the code of the class.

The API of the PingPongModel can be found at the URL `www.eecs.yorku.ca/course_archive/2016-17/S/1022/api/pingpong.api/franck/pingpong/PingPongModel.html`.

API of the PingPongModel Class

Question

Why does the API not contain the attribute pingOrPong?

API of the PingPongModel Class

Question

Why does the API not contain the attribute pingOrPong?

Answer

The attribute pingOrPong is private, whereas all features in an API are public.

APIs

- Java Standard Library (JSL)
  `docs.oracle.com/javase/8/docs/api`
- Android
  `developer.android.com/reference/packages.html`

Test the PingPongModel Class

Problem

Develop a class PingPongModelTest that tests the PingPongModel class. In particular, test all public features of the class.

Test the PingPongModel Class

Problem

Develop a class PingPongModelTest that tests the PingPongModel class. In particular, test all public features of the class.

- Specification: test the public features of the PingPongModel class
- Design: your first task
- Implementation: your second task

Unit testing

A unit test is designed to test a single unit of code, for example, a method.

Such a test should be automated as much as possible; ideally, it should require no human interaction in order to run, should assess its own results, and notify the programmer only when it fails.

A class that contains unit tests is known as a test case.

The code to be tested is known as the unit under test.

JUnit

JUnit is a Java unit testing framework written by Kent Beck and Erich Gamma.

JUnit is available at `www.junit.org`.

Kent Beck

Kent Beck is an American software engineer and the creator of the Extreme Programming and Test Driven Development software development. He works at Facebook.

source: Three Rivers Institute

Erich Gamma

Erich Gamma is a Swiss computer scientist and member of the "Gang of Four" who wrote the influential software engineering textbook "Design Patterns: Elements of Reusable Object-Oriented Software." He works at Microsoft.

source: Pearson

Java annotations

Annotations provide data about code that is not part of the code itself. Therefore, it is also called metadata.

In its simplest form, an annotation looks like

```
@Deprecated
```

(The annotation type `Deprecated` is part of `java.lang` and, therefore, need not be imported.)

An annotation can include elements and their values:

```
@Test(timeout=1000)
```

(The annotation type `Test` is part of `org.junit` and, therefore, needs to be imported.)

A test case

```
import org.junit.Assert;
import org.junit.Test;

public class ...
{
    @Test
    public void ...()
    {
        ...
    }

    @Test
    public void ...()
    {
        ...
    }
}
```

Names of test methods

It is good practice to use descriptive names for the test methods.
This makes tests more readable when they are looked at later.

Assertions in test methods

Each test method should contain (at least) one assertion: an invocation of a method of the `Assert` class of the `org.unit` package.

Methods of the Assert class

`assertEquals(long, long)`

assert that the two are the same.

`assertEquals(String, long, long)`

assert that the two are the same; if not, the message is used.

Methods of the Assert class

```
assertEquals(double, double, double)
assertEquals(String, double, double, double)
```

The method invocation

```
Assert.assertEquals(expectedValue, actualValue, delta)
```

asserts

$$|\texttt{expectedValue} - \texttt{actualValue}| < \texttt{delta}$$

Methods of the Assert class

`assertEquals(Object, Object)`
`assertEquals(String, Object, Object)`

asserts that the objects are equal according to the `equals` method.

`assertSame(Object, Object)`
`assertSame(String, Object, Object)`

asserts that the objects are equal according to the == operator.

Methods of the Assert class

`assertTrue(boolean)`
`assertTrue(String, boolean)`

asserts that the condition is true.

`assertFalse(boolean)`
`assertFalse(String, boolean)`

asserts that the condition is false.

Methods of the Assert class

`assertNull(Object)`
`assertNull(String, Object)`

asserts that the object is null.

`assertNotNull(Object)`
`assertNotNull(String, Object)`

asserts that the object is not null.

Timeout

Cause a test to fail if it takes longer than a specified time in milliseconds:

```
@Test(timeout=1000)
public void ...()
{
    ...
}
```

Exceptions

Cause a test to fail if a specified exception is not thrown:

```
@Test(expected=NumberFormatException.class)
public void ...()
{
    ...
}
```

Body of unit test method

1. Create some objects.
2. Invoke methods on them.
3. Check the results using a method of the `Assert` class.

JUnit in Android Studio

Each Android project already has a package that contains a sample JUnit test case. For example, the pingpong project includes the package $\mathrm{franck.pingpong}$ which contains the $\mathrm{ExampleUnitTest}$ class.

Test the PingPongModel Class

Problem

Develop a class PingPongModelTest that tests the PingPongModel class. In particular, test all public features of the class.

Test the PingPongModel Class

Problem

Develop a class PingPongModelTest that tests the PingPongModel class. In particular, test all public features of the class.

- Specification: test the public features of the PingPongModel class
- Design: your first task
- Implementation: your second task, this time using JUnit