Assignment A1

Write a class to maintain information about PhDs --- e.g. their advisor(s) and date of PhD.

Objectives in brief:
- Get used to Eclipse and writing a simple Java class
- Learn conventions for Javadoc specs, formatting code (e.g. indentation), class invariants, method preconditions
- Learn about and use JUnit testing

Important: READ CAREFULLY, including Step 7, which reviews what the assignment is graded on.

Groups. You can do the assignment with 1 other person. FORM YOUR GROUP EARLY! Use Piazza Note @5 to search for partner!

Recommended time-table for doing A1

Start A1 the day before it is due? You may be frustrated, upset, rushed because you can’t get the help you need. With 570 students, too many will be trying to get help at the last minute. Not a good educational experience. Instead, use following schedule, which gives you a day or two after each part to get help if you need it:

- 30 Oct. Spend 20 minutes reading the assignment.
- 1 Sep. Write and test Group A methods. This includes writing the Junit test procedure for the group.
- 3 Sep. Write and test Group B methods AND Group C methods.
- 5 Sep. Write and test Group D methods.
- 6 Sep. Do point 7 of the handout: Review the learning objectives and check each of the items given in point 7. Submit on the CMS.

CHECK the pinned A1 note on the Piazza every day.

Homework

1. Course website will contain classes Time and TimeTester. The body of the one-parameter constructor is not written. Write it. The one-parameter constructor is not tested in TimeTester. Write a procedure to test it.
2. Visit course website, click on Resources and then on Code Style Guidelines. Study
   1. Naming conventions
   3.3 Class invariant
   4. Code organization
   4.1 Placement of field declarations
   5. Public/private access modifiers
3. Look at slides for next lecture; bring them to next lecture

Difference between class and object

A blueprint, design, plan
A class

Can create many objects from the same plan (class). Usually, not all exactly the same.

A house built from the blueprint
An object
Overview

- An object can contain variables as well as methods. Variable in an object is called a field.
- Declare fields in the class definition. Generally, make fields private so they can't be seen from outside the class.
- May add getter methods (functions) and setter methods (procedures) to allow access to some or all fields.
- Use a new kind of method, the constructor, to initialize fields of a new object during evaluation of a new-expression.
- Create a JUnit Testing Class to save a suite of test cases.

References to text and JavaSummary.pptx

Declaration of fields: B.5-B.6 slide 12
Getter/setter methods: B.6 slide 13, 14
Constructors: B.17-B.18 slide 15
Class String: A.67-A.73
JUnit Testing Class: none slide 74-80
Overloading method names: B.21 slide 22

class Time

Object contains the time of day in hours and minutes.
Methods in object refer to fields in object.
Could have an array of such objects to list the times at which classes start at Cornell.
With variables t1 and t2 below,
t1.getHour() is 8
t2.getHour() is 9
t2.toString() is "09:05"

/** An instance maintains a time of day */
public class Time {
    private int hr;  // hour of the day, in 0..23
    private int min; // minute of the hour, in 0..59

    /** Get hour of the day */
    public int getHour() {
        return hr;
    }

    /** Get minute of the hour */
    public int getMin() {
        return min;
    }

    /** return time as string */
    public String toString() {
        // code
    }
}

Class invariant

/** An instance maintains a time of day */
public class Time {
    private int hr;  // hour of the day, in 0..23
    private int min; // minute of the hour, in 0..59

    // Software engineering principle: Always write a clear, precise class invariant, which describes all fields.
    // Call of every method starts with class invariant true and should end with class invariant true.
    // Frequent reference to class invariant while programming can prevent mistakes.
}

Getter methods (functions)

/** An instance maintains a time of day */
public class Time {
    private int hr;  // hour of the day, in 0..23
    private int min; // minute of the hour, in 0..59

    /** Return hour of the day */
    public int getHour() {
        return hr;
    }

    /** Return minute of the hour */
    public int getMin() {
        return min;
    }
}

References to text and JavaSummary.pptx

Declaration of fields: B.5-B.6 slide 12
Getter/setter methods: B.6 slide 13, 14
Constructors: B.17-B.18 slide 15
Class String: A.67-A.73
JUnit Testing Class: none slide 74-80
Overloading method names: B.21 slide 22
A little about type (class) String

```java
public class Time {
    private int hr; // hour of the day, in 0..23
    private int min; // minute of the hour, in 0..59
    /** Return a representation of this time, e.g. 09:05*/
    public String toString() {
        return prepend(hr) + " : " + prepend(min);
    }
    /** Return i with preceding 0, if necessary, to make two chars. */
    private String prepend(int i) {
        if (i > 9 || i < 0) return "" + i;
        return "0" + i;
    }
    ...
}
```

**Setter methods (procedures)**

```java
/** An instance maintains a time of day */
public class Time {
    private int hr; // hour of the day, in 0..23
    private int min; // minute of the hour, in 0..59
    ... // omitted
    /** Change this object's hour to h */
    public void setHour(int h) {
        hr = h;
    }
    getHour() getMin() toString()
    // hr, min are now in the object
}
```
Test setter method in JUnit testing class

```
public class TimeTester {
    ...
}

@Test
public void testSetters() {
    Time t1 = new Time();
    t1.setHour(21);
    assertEquals(21, t1.getHour());
}
```

Constructors —new kind of method

```
public class Time {
    private int hr; // hour of day, 0..23
    private int min; // minute of hour, 0..59
    /** Constructor: an instance with h hours and m minutes. 
     * Precondition: h in 0..23, m in 0..59 */
    public Time(int h, int m) {
        hr = h;
        min = m;
    }
}
```

How to test a constructor

```
public class TimeTester {
    @Test
    public void testConstructor1() {
        Time t1 = new Time(9, 5);
        assertEquals(9, t1.getHour());
        assertEquals(5, t1.getMin);
    }
    ...
}
```

Revisit the new-expression

```
Syntax of new-expression: new <constructor-call>

Example: new Time(9, 5)
```

Constructors —new kind of method

```
public class C {
    private int a;
    private int b;
    private int c;
    private int d;
    private int e;
    ...
}
```

A second constructor

```
/** An object maintains a time of day */
public class Time {
    private int hr; // hour of day, 0..23
    private int min; // minute of hour, 0..59
    /** Constructor: an instance with m minutes. 
     * Precondition: m in 0..(23*60 +59) */
    public Time(int m) {
        hr = m/60;
        min = m%60;
        ??? What do we put here ???
    }
    ...
}
```

Purpose of constructor: Initialize fields of a new object so that its class invariant is true

No return type or void

Name of constructor is the class name
Generate javadoc

- With project selected in Package explorer, use menu item Project -> Generate javadoc
- In Package Explorer, click on the project -> doc -> index.html
- You get a pane with an API like specification of class Time, in which javadoc comments (start with /**) have been extracted!
- That is how the API specs were created.

Method specs should not mention fields

```java
class Time {
    private int hr; // in 0..23
    private int min; // in 0..59
    /** return hour of day*/
    public int getHour() {
        return hr;
    }
}
```

Specs of methods stay the same.
Implementations, including fields, change!