Assignment A1

Write a class to maintain information about PhDs – 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 600 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:

29 Jan. Spend 20 minutes reading the assignment.
31 Jan (Sat). Write and test the GroupA methods. This includes writing the Junit test procedure for the group.
03 Feb. Write and test the GroupB methods AND the GroupC methods.
05 Feb. Write and test the GroupD methods.
06 Feb. 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

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

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

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

    public String toString() {
        return prepend(hr) + " : " + prepend(min);
    }
}
```

Java: double quotes for String literals
Java: + is String concatenation
Concatenate with empty String to change any value to a String
"helper" function is private, so it can't be seen outside class

Class invariant

```java
public class Time {
    private int hr; // hour of the day, in 0..23
    private int min; // minute of the hour, in 0..59

    /** An instance maintains a time of day */
    class invariant: collection of defs of variables and constraints on them (green stuff)

    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.
}
```

Class Time

```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

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

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

    /** Return a representation of this time, e.g. 09:05*/
    public String toString() {
        return prepend(hr) + " : " + prepend(min);
    }
}
```

Getter methods (functions)

```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

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

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

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 i with preceding 0, if necessary, to make two chars. */
    private String prepend(int i) {
        if (i > 9 || i < 0) return "";
        return "0" + i;
    }

    public String toString() {
        return prepend(hr) + " : " + prepend(min);
    }
}
```

Object contains the time of day in hours and minutes.
Methods in object refer to field 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"

Spec goes before method. It's a Javadoc comment —starts with /**

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.
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
    ...
    /** Change this object's hour to h */
    public void setHour(int h) {
        hr = h;
    }
}
```

Test using a JUnit testing class

```java
import static org.junit.Assert.*;
import org.junit.Test;
public class TimeTester {
    @Test
    public void test() {
        Time t1 = new Time();
        assertEquals(0, t1.getHour());
        assertEquals(0, t1.getMin());
        assertEquals("00:00", t1.toString());
    }
}
```

Test setter method in JUnit testing class

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

Constructors — new kind of method

```java
public class C {
    private int a;
    private int b;
    private int c;
    private int d;
    private int e;
    ...
    public class C {
        private int a;
        private int b;
        private int c;
        private int d;
        private int e;
        ...
        C var = new C(2, 20, 35, -15, 150);
    }
    ...
    public class Time {
        private int hr; // hour of the day, in 0..23
        private int min; // minute of the hour, in 0..59
        ...
        public class Time {
            private int hr; // hour of the day, in 0..23
            private int min; // minute of the hour, in 0..59
        }
```
Constructors — new kind of method

```java
/** 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 h hours and m minutes. 
Precondition: h in 0..23, m in 0..59 */
public Time(int h, int m) {
hr = h;
min = m;
}
```