Assignment A1 is on the CMS and Piazza

Write a simple class to maintain information about bees.
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.

Homework

1. Course website contains 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
Overloading method names: B-21 slide 22

class Time

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"

```java
Time@150
hr 8
min 0
gethour() getmin() toString()

Time@fa8
hr 9
min s
gethour() getMin() toString()
```
**A 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
}
```

Access modifier `private`: can't see field from outside class
Software engineering principle: make fields private, unless there is a real reason to make public

**Class invariant**

```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
}
```

**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)**

```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;
    } // gets before method.
    /** Return minute of the hour */
    public int getMin() {
        return hr;
    }
}
```

**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;
    }
}
```

Set `hr` is now in the object

**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);
    }
}
```

Java: `+` is String concatenation

**Helper function is private, so it can't be seen outside class**

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

**Do not say “set field hr to h”**

User does not know there is a field. All user knows is that
Time maintains hours and minutes. Later, we show an implementation that
doesn't have field hr but “behavior” is the same.

Java: double quotes for String literals
Test using a Junit testing class

In Eclipse, use menu item File → New → Junit Test Case to create a class that looks like this:

```java
import static org.junit.Assert.*;
import org.junit.Test;

public class TimeTester {
    @Test
    public void test() {
        fail("Not yet implemented");
    }
}
```

Select TimeTester in Package Explorer. Use menu item Run → Run. Procedure test is called, and the call fail(...) causes execution to fail:

Write and save a suite of "test cases" in TimeTester, to test that all methods in Time are correct.

```java
public class Time Tester {
    private int a;
    private int b;
    private int c;
    private int d;
    private int e;

    public Time(float a, float b, float c, float d, float e) {
        this.a = (int) a;
        this.b = (int) b;
        this.c = (int) c;
        this.d = (int) d;
        this.e = (int) e;
    }
}
```

C has lots of fields. Initializing an object can be a pain — assuming there are suitable setter methods...

```java
C var = new C();
var.setA(2);
var.setB(20);
var.setC(35);
var.setD(-15);
var.setE(150);
```

But first, must write a new method called a constructor.

Revisit the new-expression

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

Example: new Time(9, 5)

Evaluation of new-expression:
1. Create a new object of class, with default values in fields
2. Execute the constructor-call
3. Give as value of the expression the name of the new object

If you do not declare a constructor, Java puts in this one:

```java
public <class-name> () {
```

How to test a constructor

Create an object using the constructor. Then check that all fields are properly initialized—even those that are not given values in the constructor call.

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

Note: This also checks the getter methods! No need to check them separately. But, main purpose: check constructor.

Method specs should not mention fields

```java
public 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.

A second constructor

```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 m minutes. Precondition: m in 0..23*60 + 59 */
    public Time(int m) {
        ??? What do we put here ???
    }
    ...
    new Time(9, 5)
    new Time(125)
}
```

Time is overloaded: 2 constructors! Have different parameter types. Constructor call determines which one is called.

```java
Time@fa8
hr 9 min 5
getHour() getMin()
toString() setHour(int)
Time(int, int) Time (int)
```