Assignment A1 is on the CMS and Piazza

Write a simple class to maintain information about butterflies
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.

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

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

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.getHours() is 8
t2.getHours() is 9
t2.toString() is "09:05"

### 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

**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.

### 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 hr;
    }
}
```

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

### 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;
    }
}
```

No way to store value in a field! We can add a "setter method"
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:

```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());
    }
}
```

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

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

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(-15);
var.setD(150);
```

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

```java
C = new C(2, 20, 35, -15, 150);
```

Easier way to initialize the fields, in the new-expression itself. Use:

```java
C = new C(2, 20, 35, -15, 150);
```

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

No return type or void

Name of constructor is the class name

Memorize!
**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 void getHour() {
        return h;
    }
}
```

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

**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. */
    public Time(int m) {
        ??? What do we put here ???
    }
}

New constructor:

```java
new Time(9, 5)
new Time(125)
```