Big Red Sports Network (BRSN) is Cornell's only student-run Broadcast Media, and Journalism organization dedicated entirely to Sports, with strong partnerships with Cornell Athletics, WVBR FM 93.5, and the Ithaca Voice.

UPCOMING INFO SESSIONS
WEDNESDAY 9/3: Kennedy 326 (5:00 PM)
MONDAY 9/8: Goldwin Smith 342 (5:00 PM)
THURSDAY 9/11: Goldwin Smith G76 (5:00 PM)

www.bigredsportsnetwork.org Facebook.com/cornellbrsn @CornellBRSN

CS2110 Announcements

A0
- Some people have not submitted A0 through CMS (perhaps because they had trouble getting Java/Eclipse going on their computer, or they added the class late, not in CMS, etc.).
- No late penalty (this time) for A0 handed in through Wednesday.

A1
- Will be available on CMS and the course website this morning

Piazza
- Check course Piazza regularly for announcements
- Also to learn about issues with assignments, to find partners, etc.

Assignment A1 is on course website, CMS

Write a simple class to maintain information about elephants.
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.
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

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

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

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;
    }

    /** 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 prepend(min);
        return prepend(i);
    }

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

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

/** 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;
    }

    getHour()
    getMin()
    toString()
}

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:

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

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

But first, must write a new method called a constructor
```
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;
    }
}
```

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

Memorize!

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);
    }
}
```

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 ???
    }
}
```

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

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!