CS2110 Announcements

Take course S/U?
OK with us. Check with your advisor/major. To get an S, you need to do at least C– work. Do D+ work or less, you get a U.

HW1 due on 4 September. See Piazza note @14
Please don’t email us about prelim conflicts! We’ll tell you at the appropriate time how we handle them.
If you are new to the course and want to submit a quiz or assignment that is past due, talk to or email you TA and ask for an extension.
Profs eat lunch with 7 students. Sign up on pinned Piazza note @15 to take part.
Quiz 1 doesn’t count. Later quizzes will count.
Do a recitation in groups of 1, 2, 3 in the same recitation section. Doesn’t make sense to do it with someone not in same section.

Assignment A1

Write a class to maintain information about PhDs — e.g. their advisor(s) and date of PhD. Pay attention today, you will do exactly what I do in creating and testing a class!
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 8, which reviews what the assignment is graded on.
Groups. You can do A1 with 1 other person. FORM YOUR GROUP EARLY! Use pinned Piazza Note @5 to search for partner!

Homework (not to be handed in)

1. Course website will contain classes Time and TimeTest. The body of the one-parameter constructor is not written. Write it. The one-parameter constructor is not tested in TimeTest. 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

How to learn Java syntax

Question on the course Piazza:
I worked on recitation 1 in the recitation section today, but I am still confused as to when/when not to add semicolons. Is there a general rule regarding semicolon placement in java?
Answer: Any basic statement (one that doesn’t include other statement) require ; at end, e.g.
   assignment
   return
   procedure call

How to learn Java syntax

When you have a question on syntax of statements, there are two ways to find a suitable answer:
1. Try it in Eclipse — keep trying different things until something works. HORRIBLE. You waste your time and learn nothing.
2. Look up the statement in JavaHyperText! Wonderful! Look up a statement twice and you will know it forever.
## 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, run them when necessary.

## References in JavaHyperText entries

- Look at these JavaHyperText entries:
  - Declaration of fields: field
  - Getter/setter methods: getter setter
  - Constructors: constructor
  - Class String: toString
  - JUnit Testing Class: Junit
  - Overloading method names: overload
  - Overriding method names: override

## 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

    Time() {
    }
}
```

## 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

    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.

    Time() {
    }
}
```
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
private String hr = "9";
private String min = "5";
```

### 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
public class TimeTest {
    @Test
    public void test() {
        fail("Not yet implemented");
    }
}
```
Test using a JUnit testing class

```java
public class TimeTest {
    @Test
    public void testConstructor() {
        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 TimeTest, to test that all methods in Time are correct.

Test setter method in JUnit testing class

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

TimeTest can have several test methods, each preceded by @Test. All are called when menu item Run ➞ Run is selected.

Constructors — new kind of method

```java
public class C {
    private int a;
    private int b;
    private int c;
    private int d;
    private int e;
    C var = new C();
    var.setA(2);
    var.setB(20);
    var.setC(35);
    var.setD(-15);
    var.setE(150);
}
```

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

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

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

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

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

**An object maintains a time of day**

```java
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. */
    public Time(int h, int m) {
        hr = h;
        min = m;
    }
}
```

Precondition: h in 0..23, m in 0..59

Need precondition

**No return type or void**
Name of constructor is the class name

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

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

Revisit the new-expression

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

```java
Time t1 = 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:
```
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
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
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;
    }
    ...
}

Method specs should not mention fields

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

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.

Next week's section: Exception Handling

Suppose we are supposed to read an integer from the keyboard
and do something with it. If the user types something other than
an integer, we want to ask the user again to type a integer.

String st= the integer from the keyboard;
int k=Integer.parseInt(st); // return the int that is in st

public static int parseInt(String s) {
    ...
    user typed “x13”,
    it was discovered here
    parseInt doesn’t know what to do with the error
    NFE@2

You must read/watch the tutorial BEFORE the recitation:
Look at the pinned Piazza note Recitation/Homework.
Quiz on tutorial Monday beginning at 3PM. Bring your laptop to
class, ready to solve problems (write code).
During the section, you can talk to neighbors, discuss things,
answer questions together. The TA will walk around and help. The
TA will give a short presentation on some issue if needed.
You will have until Friday after the recitation to submit answers on
the CMS. But best to complete and submit during recitation.
Do it in groups of 1, 2, 3 people in same section. Form group on
Piazza BEFORE submitting. One must invite, others must accept.