1

CS/ENGRD 2110
SPRING 2016

Lecture 3: Fields, getters and setters, constructors, testing
http://courses.cs.cornell.edu/cs2110

CS2110 Announcements

2

A1 now on course website and Piazza.

Piazza

☐ Check course Piazza regularly for announcements.

☐ To learn about issues with A1, check the pinned A1 FAQs (Frequently Asked Questions) note often!

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.

VideoNote.com.

Last semester's videos are under 2016! Don't look for fall 2015.

Assignment A1

3

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

4

Start A1 the day before it is due? You may be frustrated, upset, rushed because you can't get the help you need. With 500 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:

4 Feb. Spend 20 minutes reading the assignment.

6 Feb. Write and test the Group A methods. This includes writing the Junit test procedure for the group.

7 Feb. Write and test the Group B methods AND the Group C methods.

8 Feb. Write and test the GroupD methods.11

10 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

5

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

6

☐ 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

7

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

8

Object contains the time of day in hours and minutes.

Methods in object refer to fields 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"

Class Time

9

/** 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

10

/** 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.

}

Getter methods (functions)

11

/** 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;

   }

}

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

A little about type (class) String

12

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 represention of this time, e.g. 09:05*/

   public String toString() {

      return prepend(hr) + ":" + prepend(min);

   }

   /** Return i with preceding 0, if
      necessary, to make two chars. */

   private String prepend(**int** i) {

      if (i > 9 || i < 0) **return** "" + i;

      return "0" + i;

   }

   …

Java: double quotes for String literals

Java: + is String catenation

Catenate with empty String to change any value to a String

"helper" function is private, so it can't be seen outside class

Concatenate or catenate?

13

I never **concatenate** strings;
I just **catenate** those little things.
Of syllables few,
I'm a man through and through.
Shorter words? My heart joyfully sings!

Setter methods (procedures)

14

```
/** 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"

Time@fa8
hr 9 | Time
min 5 | getHour()
getMin()
setHour(int) | toString()

setHour(int) is now in the object

Setter methods (procedures)

15

```
/** An instance maintains a time of day */
public class Time {
    private int hr;   //hour of day, in 0..23
    private int min; // minute of hour, in 0..59
    …
    /** Change this object's hour to h */
    public void setHour(int h) {
        hr= h;
    }

}
```

Time@fa8
hr 9 | Time
min 5 | getHour()
getMin()
setHour(int) | toString()

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 h but "behavior" is the same

---

Test using a JUnit testing class

16

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

```
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:

Runs: 1/1   Errors: 0   Failures: 1

▼ TimeTester [Runner: JUnit 4] (0.001 s)
   test (0.001 s)

Test using a JUnit testing class

17

```
…
public class TimeTester {
    @Test
    public void testConstructor() {
        Time t1= new Time();
        assertEquals(0, t1.getHour());
        assertEquals(0, t1.getMin();
        assertEquals("00:00", t1.toString());
    }
}
```

Store new Time object in t1.

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

Give green light if expected value equals computed value, red light if not:
assertEquals(expected value, computed value);

Test setter method in JUnit testing class

18

```
public class TimeTester {
    …

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

TimeTester can have several test methods, each preceded by @Test.

All are called when menu item Run→ Run is selected

Time@fa8
hr 9 | Time
min 5 | getHour()
getMin()
setHour(int) | toString()

Constructors —new kind of method

19

```java
public class C {
    private int a;
    private int b;
    private int c;
    private int d;
    private int e;
}
```

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

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

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

Constructors —new kind of method

20

```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 fields of a new object so that its class invariant is true

Memorize!

Need precondition

No return type or void

Name of constructor is the class name

Time@fa8

| hr | 9 | min | 5 | Time |

getHour() getMin()
toString() setHour(int)
Time(int, int)

Revisit the new-expression

21

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:
public <class-name> () { }

Time@fa8

| hr | 9 | min | 5 | Time |

getHour() getMin()
toString() setHour(int)
Time(int, int)

How to test a constructor

22

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

A second constructor

23

```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) {
        hr= m/60; min= m%60;
        ??? 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

Time@fa8

| hr | 9 | min | 5 | Time |

getHour() getMin()
toString() setHour(int)
Time(int, int) Time (int)

Generate javadoc

24

□ 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.

Method specs should not mention fields