

CS2110 Announcements

Δ

- 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.).
- $\hfill\Box$ 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

4

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
- $\hfill\Box$ Learn about and use JUnit testing

Important: read carefully, including Step 7, which reviews what the assignment is graded on.

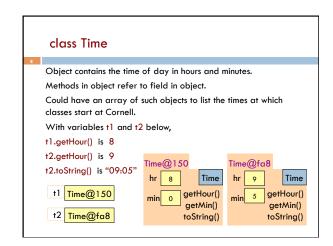
Homework

- 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



```
/** 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

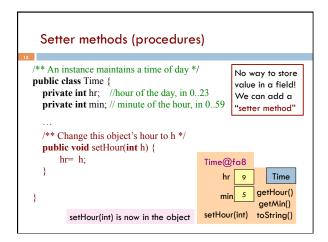
Access modifier private:
    can't see field from outside class
    Software engineering principle:
    make fields private, unless there
    is a real reason to make public

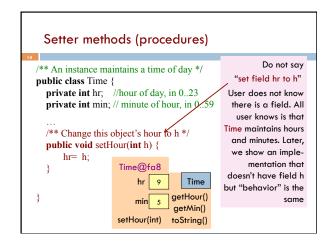
}
```

```
Class invariant
                                                 Class invariant:
                                                 collection of defs of
/** An instance maintains a time of day */
                                                 variables and
                                                 constraints on them
public class Time {
                                                (green stuff)
  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)
/** 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 */=
                                      Spec goes before method.
  public int getHour() {
                                      It's a Javadoc comment
    return hr;
                                      -starts with /**
                                              Time@fa8
  /** Return minute of the hour */
                                                  9
                                                           Time
  public int getMin() {
    return min;
                                                   5
                                                       getHour()
                                                       getMin()
```

```
A little about type (class) String
public class Time {
  private int hr; //hour of the day, in 0..23
                                                       Java: double
  private int min; // minute of the hour, in 0..59
                                                          quotes for
  /** Return a represention of this time, e.g. 09:05*
                                                       String literals
  public String toString() {
    return prepend(hr) + ":" + prepend(min);
                                                           Java: + is
                                                               String
  /** Return i with preceding 0, if
                                                          catenation
     necessary, to make two chars. */
  private String prepend(int i) {
                                  Concatenate with empty String
    if (i > 9 || i < 0) return "" + i;
                                  to change any value to a String
    return "0" + i;
                                  "helper" function is private, so it
                                  can't be seen outside class
```



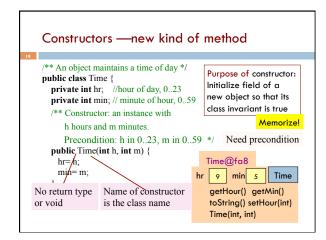


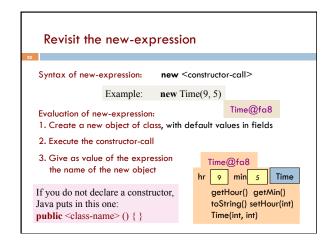
```
Test using a JUnit testing class
In Eclipse, use menu item File → New → JUnit Test Case to
create a class that looks like this:
                                       Select TimeTester in Package
import static org.junit.Assert.*;
                                       Explorer.
import org.junit.Test;
public class TimeTester {
                                       Use menu item Run → Run.
  public void test() {
                                       Procedure test is called, and
     fail("Not yet implemented");
                                       the call fail(...) causes
                                       execution to fail:
                                  Runs: 1/1 Errors: 0 E Failures: 1
                                 TimeTester [Runner: JUnit 4] (0.001 s)
```

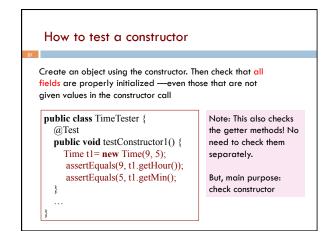
```
Test using a JUnit testing class
                                        Write and save a suite of
                                        "test cases" in TimeTester,
public class TimeTester {
                                       to test that all methods in
  @Test
                                        Time are correct
  public void test() {
                               Store new Time object in t1.
     Time t1 = new Time();
     assertEquals(0, t1.getHour());
     assertEquals(0, t1.getMin();
     assertEquals("00:00", t1.toString());
         Give green light if expected value equals
              computed value, red light if not:
         assertEquals(expected value, computed value);
```

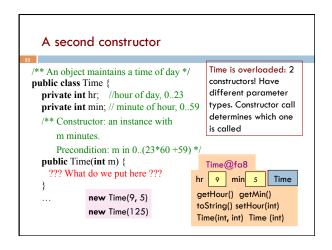
```
Test setter method in JUnit testing class
public class TimeTester {
                                      TimeTester can have
                                      several test methods, each
                                      preceded by @Test.
  @Test
                                      All are called when menu
  public void testSetters() {
                                      item Run→ Run is selected
     Time t1= new Time();
     t1.setHour(21):
                                         Time@fa8
     assertEquals(21, t1.getHour());
                                                        Time
  }
                                                    getHour()
                                                5
                                                     getMin()
                                        setHour(int) toString()
```

```
Constructors —new kind of method
public class C {
                        C has lots of fields. Initializing an
 private int a;
                         object can be a pain —assuming
 private int b;
                         there are suitable setter methods
 private int c;
 private int d;
                        Easier way to initialize the fields, in
 private int e;
                        the new-expression itself. Use:
C var= new C();
                           C var= new C(2, 20, 35, -15, 150);
var.setA(2);
var.setB(20);
                         But first, must write a new method
var.setC(35);
                         called a constructor
var.setD(-15);
var.setE(150);
```









```
Method specs should not mention fields
public class Time {
                                       public class Time {
                                          // min, in 0..23*60+59
 private int hr; //in 0..23
 private int min; //in 0..59
                                          private int min;
  /** return hour of day*/
                                          /** return hour of day*/
                           to change
 public int getHour() {
                                          public int getHour() {
                                              return min / 60;
    return h;
                                             Time@fa8
  Time@fa8
                                           min 545
                                                          Time
                  Time
      9
                                           getHour() getMin()
                                           toString() setHour(int)
              getHour()
  min
      5
               getMin()
                         Specs of methods stay the same.
              toString()
 setHour(int)
                         Implementations, including fields, change!
```