CS/ENGRD 2110 SPRING 2019

Lecture 3: Fields, getters and setters, constructors, testing

http://courses.cs.cornell.edu/cs2110

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 29 January. See Piazza note @22.

A0 due on 30 January. See Piazza note @23.

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 @8 to take part.

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.

CS2110 Grading HW1: Comments from you

We *started* grading. The 15 that I looked at got it right, although a few could be better worded, and we say that in the feedback.

These tasks seemed trivial, but after completion I see their importance. These activities helped develop a solid foundation of good programing and understanding algorithms.

I thought the two videos were very helpful. I was definitely confused about how to answer the question at the end of class. It made me realize how important semantics and the choice of words is. It has made me more careful about my choice of words .

… the chef/recipe analogy was really helpful …

Interesting information/exercise. I programmed java before but it is quite helpful to abstract away the details of programming to yield what is "really" going on, and feel this will be especially helpful as we move on to more complex programs/topics.

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 9, 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

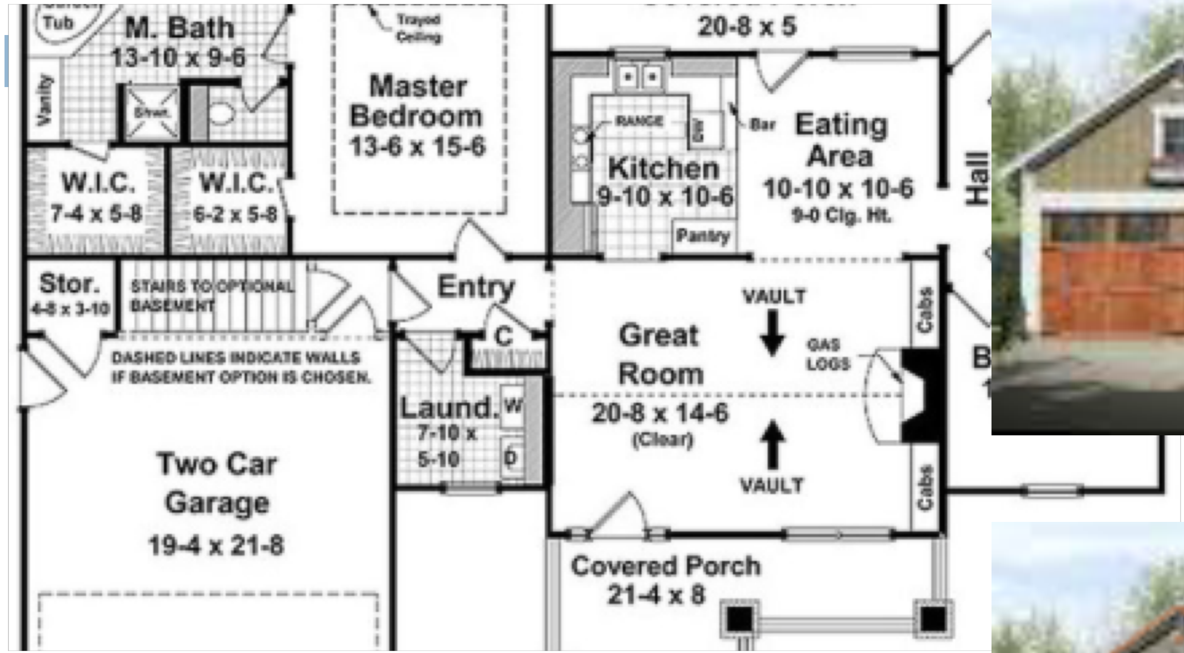
procedure call

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:

Class definition: classes

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

Object contains the time of day in hours and minutes.

Methods in object refer to fields in object.

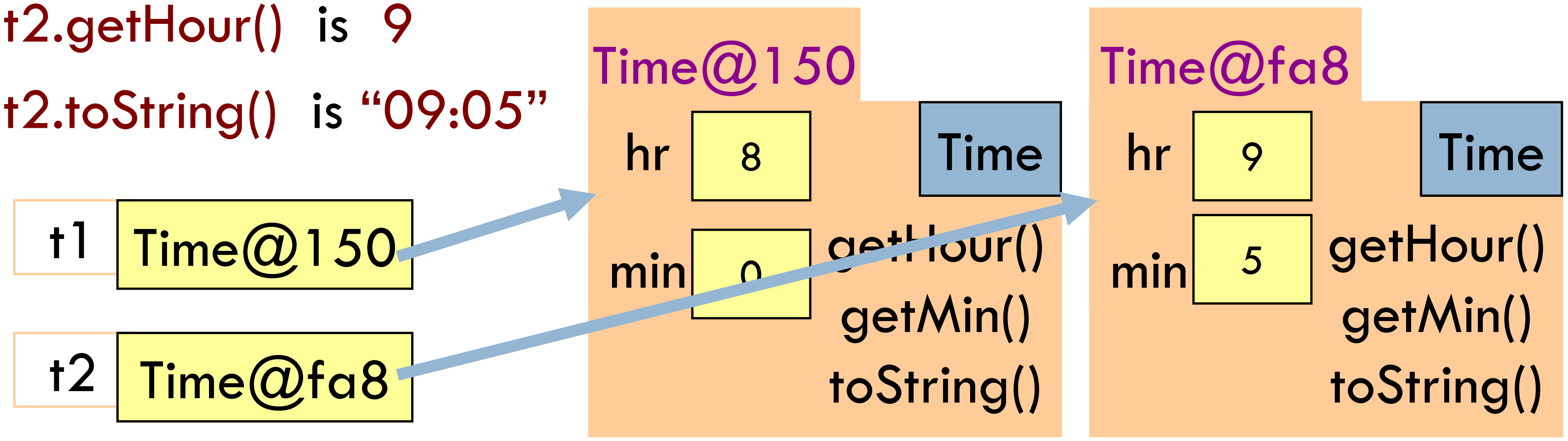
Could have an array of such objects to list the times at which classes start at Cornell.

With variables t1 and t2 below,

t1.getHour() is 8

t2.getHour() is 9

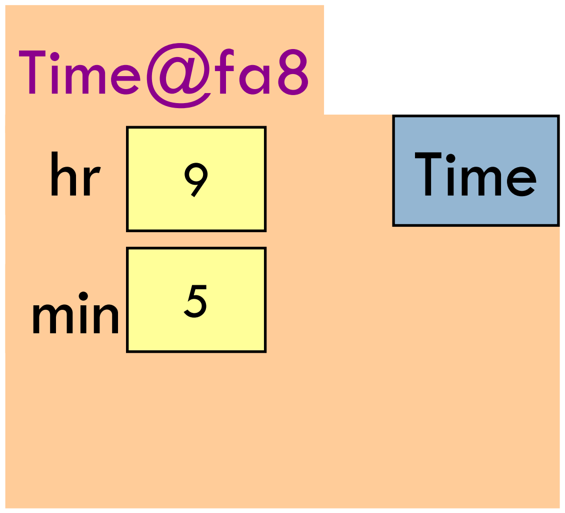
t2.toString() is "09:05"



Class Time

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

```
/** An instance maintains a time of day */
public class Time {
    /** hour of the day, in 0..23. */
    private int hr;
    /** minute of the hour, in 0..59. */
    private int min;
```



Class Time

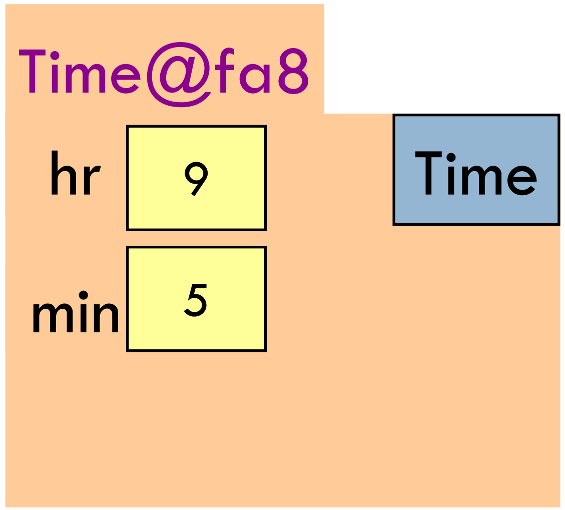
Class invariant: collection of defs of variables and constraints on them (blue stuff)

Software engineering principle: Always write a clear, precise class invariant.

Every method call starts with class inv true and should end with class inv true.

Frequent reference to class inv can prevent mistakes.

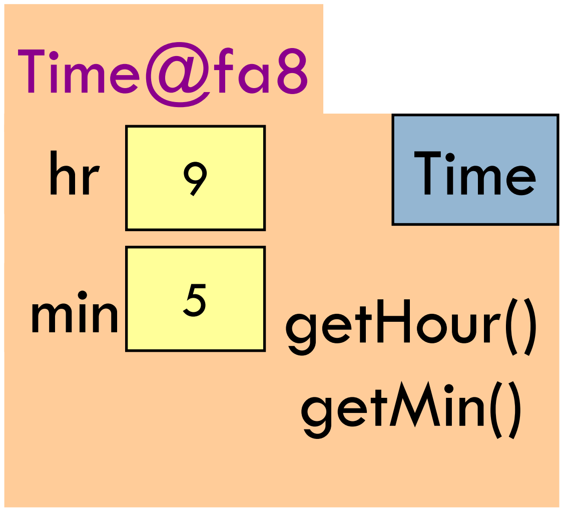
```
/** An instance maintains a time of day */
public class Time {
    /** hour of the day, in 0..23. */
    private int hr;
    /** minute of the hour, in 0..59. */
    private int min;
```



Getter methods (functions)

```
public class Time {
   /** Hour of the day, in 0..23. */
   private int hr;
   /** Minute of the hour, in 0..59 */
  private int min;

   /** 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

```
public class Time {
   private int hr;
   private int min;
   /** = 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?

I never **concatenate** strings;

I just **catenate** those little things.

Of syllables few,

I'm a man through and through.

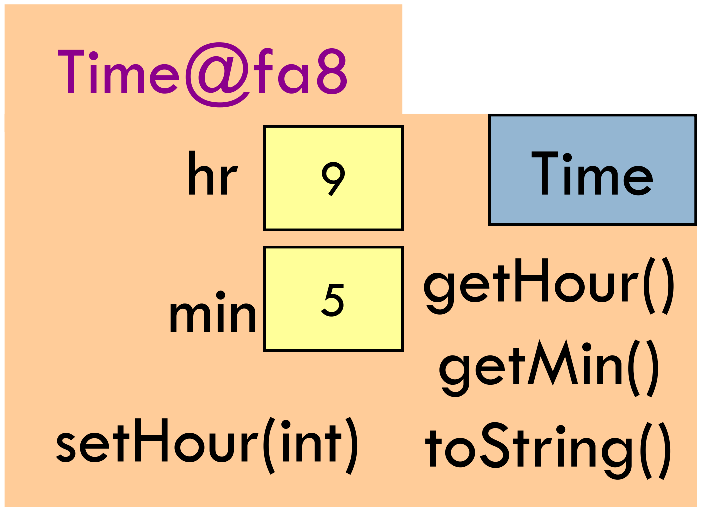
Shorter words? My heart joyfully sings!

Setter methods (procedures)

```
/** An instance maintains a time of day */
public class Time {
    private int hr;  // in 0..23
    private int min; // in 0..59


    /** Change this object's hour to h.
      *   Precondition:  h in 0..23. */
    public void setHour(int h) {
        hr= h;
}  }
```

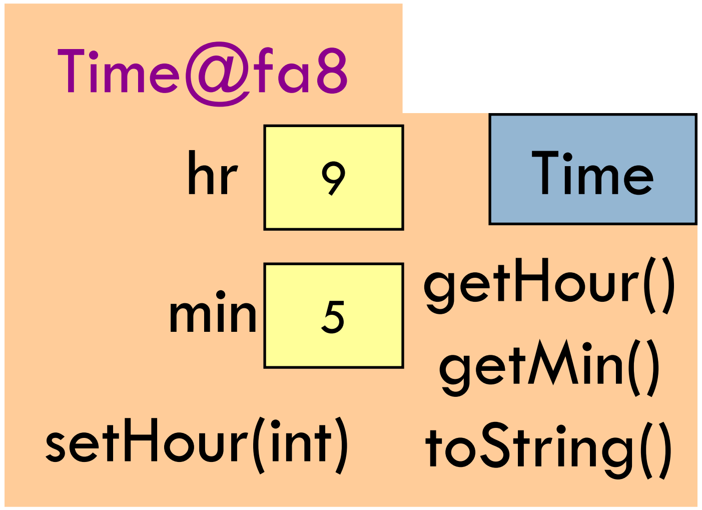
No way to store value in a field! We can add a "setter method"

setHour(int) is now in the object



Setter methods (procedures)

```
/** An instance maintains a time of day */
public class Time {
   private int hr;
   private int min;

  /** Change this object's hour to h.
    *   Precondition:  h in 0..23. */
  public void setHour(int h) {
       hr=  h;
  }

}
```



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 imple-mentation that doesn't have field h but "behavior" is the same

Test using a JUnit testing class

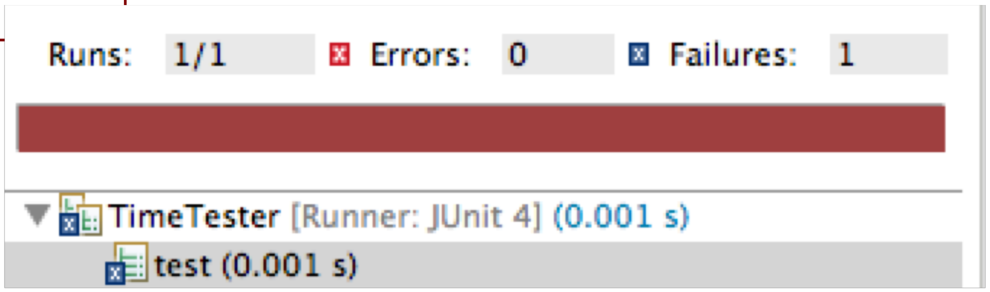
In Eclipse, use menu item File → New → JUnit Test Case to create a class that looks like this:

```
import static org.junit.Assert.*;
import org.junit.Test;

public class TimeTest {
   @Test
   public void test() {
      fail("Not yet implemented");
   }
}
```

Select TimeTest 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

```
…
public class TimeTest {
   @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 TimeTest, to test that all methods in Time are correct

Store new Time object in t1.

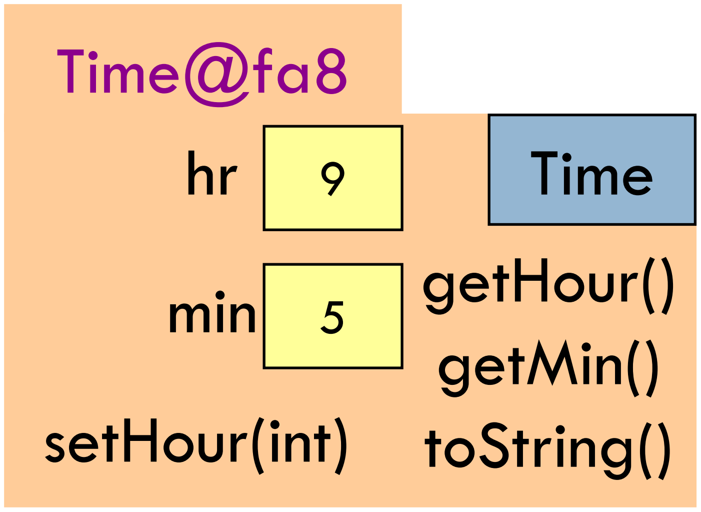
Give green light if expected value equals computed value, red light if not:
assertEquals(expected value, computed value);

Test setter method in JUnit testing class

```
public class 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

```
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:

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

But first, must write a new method called a constructor

Constructors —new kind of method

```
/** 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;
    }
```

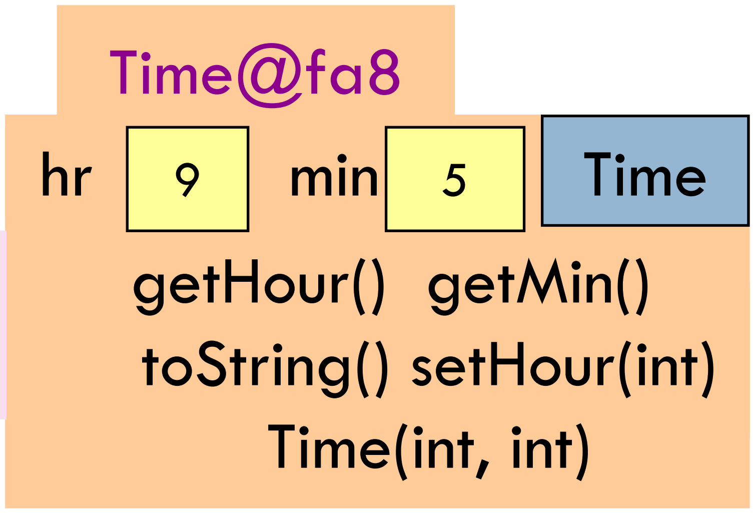
Purpose of constructor: Initialize fields of a new object so that its class invariant is true

Memorize!

Need precondition

No return type or void

Name of constructor is the class name



Revisit the new-expression

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

Example: **new** Time(9, 5)

Time@fa8

Evaluation of new-expression:

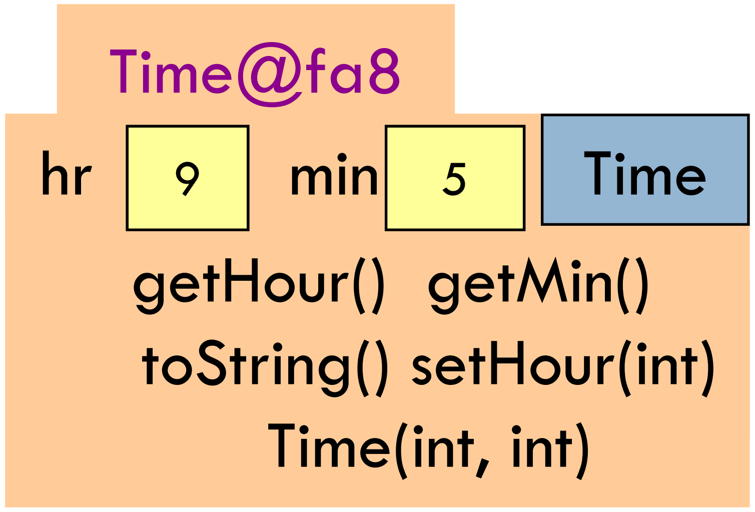
1. Create a new object of class, with default values in fields

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

```
public class TimeTest {
    @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

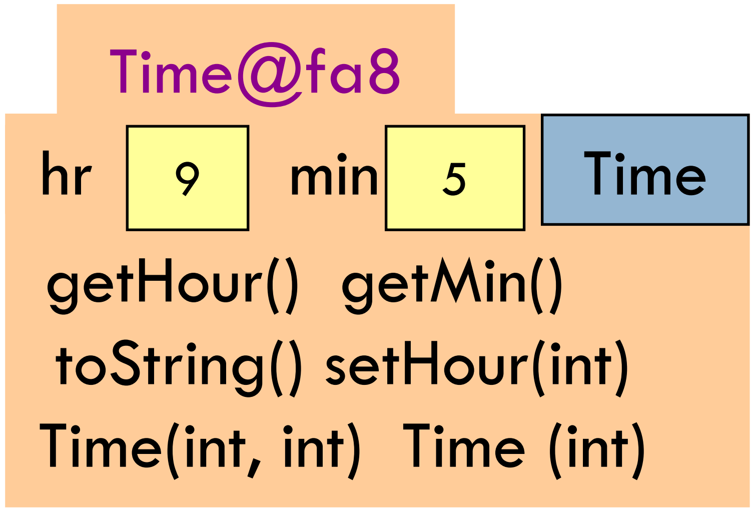
A second constructor

```
/** 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 ???
    }
    …
```

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

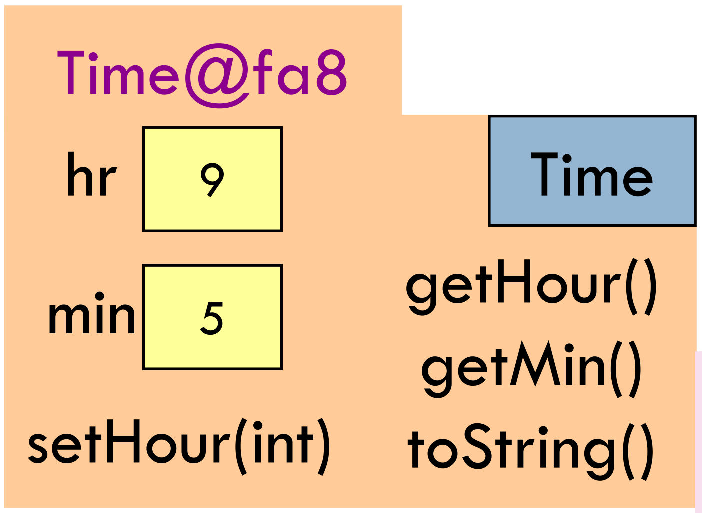
new Time(9, 5)

new Time(125)



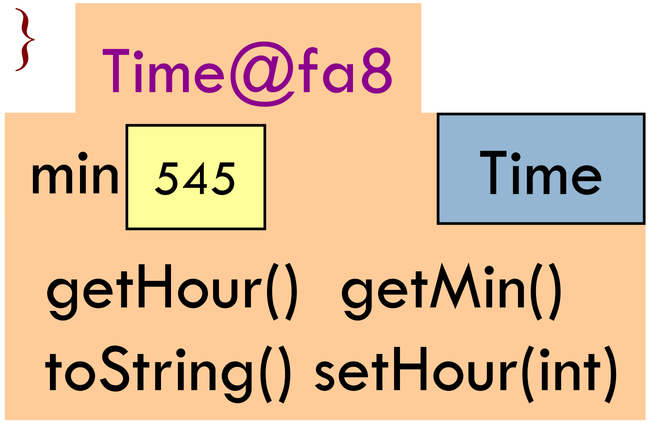
Method specs should not mention fields

```
public class Time {
  private int hr;
  private int min;

  /** return hour of day*/
  public int getHour() {
     return h;
  }
```



Decide to change implemen-tation

```
public class Time {
    /** min, in 0..23*60+59. */
    private int min;

    /** return hour of day*/
    public int getHour() {
        return min / 60;
    }
```



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