|  |
| --- |
| **Workshop 1: Simple Time** |

**Objectives**

After completing this lab, you should be able to:

* write a class definition
* use conditional statements to solve problems
* run JUnit tests to verify the correctness of your code

**Code**

Write a new class that represents the time of day:

* Create a new project (don't just stuff this code in an existing project)
* Create a new class called SimpleTime (It is important that you use the name SimpleTime.)

.

* Define three private integer instance variables to hold the hours, minutes and seconds. This will be military time where 0:00:00 is midnight and 13:30:00 is half past 1 pm.
* Implement the following methods. It is important that you name the methods exactly as shown below.
  + public SimpleTime(int h, int m, int s) - this constructor initializes the three instance variables
  + public SimpleTime( ) - this default constructor initializes the three instance variables midnight
  + public boolean isMidnight( ) - return true if the current time is 0:00:00
  + public boolean isNoon( ) - return true if the current time is 12:00:00
  + public boolean isLunchTime( ) - return true if the current time is between 12:00:00 and 13:00:00 (inclusive)
  + public String toString ( ) - use the three instance variables to build a String representation of the time. For example, 14:12:56.
  + public void increment ( ) - increases the current time by one second. Pay attention to end of rollovers of seconds and minutes.

As you complete each method, use BlueJ's object inspector to verify that your methods are correct.

When you have completed all the methods above,

* download SimpleTimeTest.java and add it to your project's folder.
* Compile all files in the project.
* Go to Tools => Testing => Run Tests to run the unit tests.
* Correct any bugs identified by the unit tests. (You may find that some of the instructions above are incomplete. Look at the source of the unit tests to determine the expected behavior. Examples can be easier to understand that long, explanations. Therefore, some companies use unit tests to partially document their code.)

**Submission**

When your code passes all of the unit tests

1. Print out your code. (One printout per team.)
2. Demonstrate to the instructor or lab assistant that your code passes all the unit tests.
3. The instructor or lab assistant will initial your printout

**Challenge Problem**

Add this method public int secondsElapsed(SimpleTime start) that will calculate the number of seconds that have elapsed between start and the time represented by the current object. Add test methods to SimpleTimeTest to verify that your code is correct.

**import** org.junit.\*;  
  
*/\*\*  
 \* Unit tests for SimpleTime  
 \*  
 \** ***@author*** *Zachary Kurmas  
 \*/***public class** SimpleTimeTest  
{  
   
 @Test  
 **public void** defaultConstructorSetsToMidnight()   
 {  
 SimpleTime time = **new** SimpleTime();  
 Assert.assertTrue(time.isMidnight());  
 }  
   
 @Test  
 **public void** isMidnightRecognizesMidnight()  
 {  
 SimpleTime time = **new** SimpleTime(0,0,0);  
 Assert.assertTrue(time.isMidnight());  
 }  
   
 @Test  
 **public void** isMidnightRecognizesNotMidnight()  
 {  
 SimpleTime time1 = **new** SimpleTime(0, 0, 1);  
 Assert.assertFalse(**"12:00:01 should be false"**, time1.isMidnight());  
   
 SimpleTime time2 = **new** SimpleTime(23, 59, 59);  
 Assert.assertFalse(**"23:59:59 should be false"**, time2.isMidnight());  
   
 SimpleTime time3 = **new** SimpleTime(12, 0, 0);  
 Assert.assertFalse(**"12:00:00 should be false"**, time3.isMidnight());  
   
 SimpleTime time4 = **new** SimpleTime(0, 1, 0);  
 Assert.assertFalse(**"0:01:00 should be false"**, time4.isMidnight());  
 }  
   
 @Test  
 **public void** isNoonRecognizesNoon()  
 {  
 SimpleTime time = **new** SimpleTime(12,0,0);  
 Assert.assertTrue(time.isNoon());  
 }  
   
 @Test  
 **public void** isNoonRecognizesNotNoon()  
 {  
 SimpleTime time1 = **new** SimpleTime(12, 0, 1);  
 Assert.assertFalse(**"12:00:01 should be false"**, time1.isNoon());  
   
 SimpleTime time2 = **new** SimpleTime(11, 59, 59);  
 Assert.assertFalse(**"11:59:59 should be false"**, time2.isNoon());  
   
 SimpleTime time3 = **new** SimpleTime(0, 0, 0);  
 Assert.assertFalse(**"0:00:00 should be false"**, time3.isNoon());  
   
 SimpleTime time4 = **new** SimpleTime(12, 1, 0);  
 Assert.assertFalse(**"12:01:00 should be false"**, time4.isNoon());  
 }  
   
   
 @Test  
 **public void** isLunchTimeRecognizesNoon()  
 {  
 SimpleTime time = **new** SimpleTime(12,0,0);  
 Assert.assertTrue(time.isLunchTime());  
 }  
   
 @Test  
 **public void** isLunchTimeRecognizesOneOClock()  
 {  
 SimpleTime time = **new** SimpleTime(13, 0 ,0);  
 Assert.assertTrue(time.isLunchTime());  
 }  
   
 @Test  
 **public void** isLunchTimeRecognizesNoonHour()  
 {  
 SimpleTime time = **new** SimpleTime(12, 0 ,1);  
 Assert.assertTrue(**"12:00:01"**, time.isLunchTime());  
   
 SimpleTime time2 = **new** SimpleTime(12, 1 ,0);  
 Assert.assertTrue(**"12:01:00"**, time2.isLunchTime());  
   
 SimpleTime time3 = **new** SimpleTime(12, 30 ,0);  
 Assert.assertTrue(**"12:30:00"**, time3.isLunchTime());  
   
 SimpleTime time4 = **new** SimpleTime(12, 59 ,0);  
 Assert.assertTrue(**"12:59:00"**, time4.isLunchTime());  
   
 SimpleTime time5 = **new** SimpleTime(12, 59 ,59);  
 Assert.assertTrue(**"12:59:59"**, time5.isLunchTime());  
 }  
   
 @Test  
 **public void** isLunchTimeNotAcceptRestofOneOClock()   
 {  
 SimpleTime time = **new** SimpleTime(13, 0 ,1);  
 Assert.assertFalse(**"13:00:01"**, time.isLunchTime());  
   
 SimpleTime time2 = **new** SimpleTime(13, 1 ,0);  
 Assert.assertFalse(**"13:01:00"**, time2.isLunchTime());  
   
 SimpleTime time3 = **new** SimpleTime(13, 59 ,0);  
 Assert.assertFalse(**"13:59:00"**, time3.isLunchTime());   
 }  
   
 @Test  
 **public void** isLunchTimeNotAcceptOtherHours()   
 {  
 SimpleTime time = **new** SimpleTime(11, 59 ,59);  
 Assert.assertFalse(**"11:59:59"**, time.isLunchTime());  
   
 SimpleTime time2 = **new** SimpleTime(14, 0 ,0);  
 Assert.assertFalse(**"14:00:00"**, time2.isLunchTime());  
   
 SimpleTime time3 = **new** SimpleTime(0, 0 ,0);  
 Assert.assertFalse(**"0:00:00"**, time3.isLunchTime());   
 }  
   
 @Test  
 **public void** toStringDoubleDigits()  
 {  
 SimpleTime time = **new** SimpleTime(10, 30, 45);  
 Assert.assertEquals(**"10:30:45"**, time.toString());  
 }  
   
 @Test  
 **public void** toStringSingleDigits()  
 {  
 SimpleTime time = **new** SimpleTime(3, 7, 9);  
 Assert.assertEquals(**"3:07:09"**, time.toString());  
 }  
   
 @Test  
 **public void** toStringMidnight()  
 {  
 SimpleTime time = **new** SimpleTime(0,0, 0);  
 Assert.assertEquals(**"0:00:00"**, time.toString());  
 }  
   
 @Test  
 **public void** toStringMixed()  
 {  
 SimpleTime time = **new** SimpleTime(7,10, 5);  
 Assert.assertEquals(**"7:10:05"**, time.toString());  
   
 SimpleTime time2 = **new** SimpleTime(10,6, 36);  
 Assert.assertEquals(**"10:06:36"**, time2.toString());  
 }  
   
 @Test  
 **public void** incrementMidnight()   
 {  
 SimpleTime time = **new** SimpleTime(0, 0, 0);  
 time.increment();  
 Assert.assertEquals(**"0:00:01"**, time.toString());   
 }  
   
 @Test  
 **public void** incrementMinuteRollover()   
 {  
 SimpleTime time = **new** SimpleTime(3, 17, 59);  
 time.increment();  
 Assert.assertEquals(**"3:18:00"**, time.toString());   
 }  
   
 @Test  
 **public void** incrementHourRollover()   
 {  
 SimpleTime time = **new** SimpleTime(5, 59, 59);  
 time.increment();  
 Assert.assertEquals(**"6:00:00"**, time.toString());   
 }  
   
 @Test  
 **public void** incrementDayRollover()   
 {  
 SimpleTime time = **new** SimpleTime(23, 59, 59);  
 time.increment();  
 Assert.assertEquals(**"0:00:00"**, time.toString());   
 }  
   
 @Test  
 **public void** multipleIncrement()   
 {  
 SimpleTime time = **new** SimpleTime(10, 58, 50);  
   
 **for** (**int** i = 0; i < 137; i++) {  
 time.increment();  
 }  
 Assert.assertEquals(**"11:01:07"**, time.toString());   
 }  
   
   
}