COMS W1007            TR 1:10-2:25 PM

Columbia University                          Fall 2014

**Homework 2: Theory**

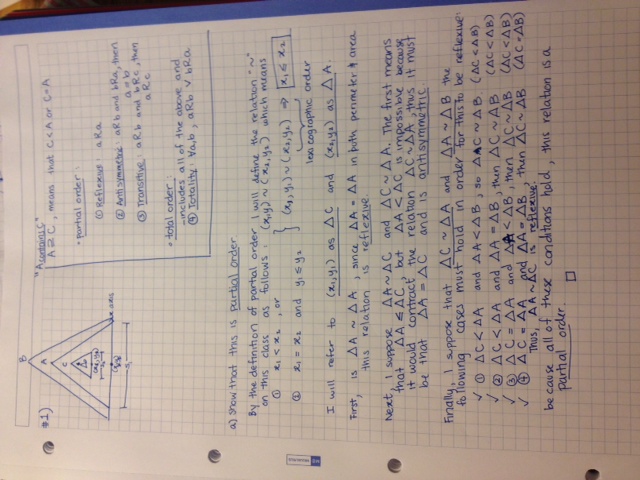
**October 14, 2014**

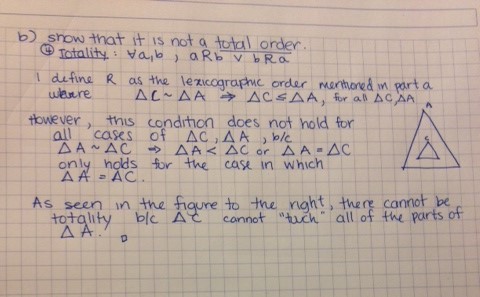
CUNIX ID: ami2119

Last Name: Iwamizu

First Name: Akiko

1) (10 points)





1. (10 points)

DaysSinceTester.java

public class DaysSinceTester

{

public static void main(String[] args)

{

System.out.print("The number of days since Amethyst Amelia Kelly's birthday: ");

DaysSince iggyBday = new DaysSince();

System.out.println(iggyBday.getDaysSince() + " days.");

}

}

DaysSince.java

import java.util.Calendar;

import java.util.GregorianCalendar;

public class DaysSince

{

private Calendar dateToday;

private final GregorianCalendar BDAY = new GregorianCalendar(1990, Calendar.JUNE, 7);

public DaysSince()

{

dateToday = new GregorianCalendar();

}

public int getDaysSince()

{

int daysSince = 0;

GregorianCalendar ameliaBday = BDAY;

int startYear = ameliaBday.get(Calendar.YEAR);

int differenceYear = dateToday.get(Calendar.YEAR) - ameliaBday.get(Calendar.YEAR);

for(int i = 0; i < differenceYear; i++)

{

daysSince += 365;

if(ameliaBday.isLeapYear(startYear))

daysSince++;

startYear++;

}

daysSince += dateToday.get(Calendar.DAY\_OF\_YEAR);

daysSince -= BDAY.get(Calendar.DAY\_OF\_YEAR);

return daysSince;

}

}

1. (15 points)

TimeOfDay.java

public class TimeOfDay

{

private int h;

private int m;

private int s;

private String d;

public TimeOfDay(int hrs, int mins, int secs, String dayTime)

{

if(1 <= hrs && hrs <= 12 && 0 <= mins && mins <= 59 && 0 <= secs && secs <= 59)

{

if(!dayTime.equals("a") && !dayTime.equals("p"))

throw new IllegalArgumentException("For a day time enter a or p");

else

{

h = hrs;

m = mins;

s = secs;

d = dayTime;

}

}

else

{

throw new IllegalArgumentException("Time entered was not valid.");

}

}

private int getHours()

{

return h;

}

private int getMinutes()

{

return m;

}

private int getSeconds()

{

return s;

}

private String getDayTime()

{

return d;

}

public TimeOfDay addSeconds(int secsAdded)

{

TimeOfDay newTime;

int newHrs = h;

int newMins = m;

int newSecs = s + secsAdded;

String newDayTime = d;

if(newSecs >= 60)

{

newMins++;

newSecs -= 60;

}

if(newMins >= 60)

{

newHrs++;

newMins -= 60;

}

if(newDayTime.equals("a"))

{

if(newHrs > 12)

newHrs -= 12;

else if(newHrs == 12)

newDayTime = "p";

}

else if(newDayTime.equals("p"))

{

if(newHrs > 12)

newHrs -= 12;

else if(newHrs == 12)

newDayTime = "a";

}

newTime = new TimeOfDay(newHrs, newMins, newSecs, newDayTime);

return newTime;

}

public int secondsFrom(TimeOfDay other)

{

int seconds, minutes, hours;

if(other.h < h)

hours = (other.h - h) \* -1;

else

hours = other.h - h;

if(other.m < m)

{

minutes = other.m + 60 - m;

hours--;

}

else

minutes = other.m - m;

if(other.s < s)

{

seconds = other.s + 60 - s;

minutes--;

}

else

seconds = other.s - s;

if(d.equals(other.d))

{

if(h > other.h)

hours = 24 - hours;

return(hours \* 3600 + minutes \* 60 + seconds);

}

else

{

if(other.d.equals("p"))

hours = 24 - hours;

else

;//hours -= 12;

return(hours \* 3600 + minutes \* 60 + seconds);

}

}

public String toString()

{

StringBuilder s = new StringBuilder();

s.append(getHours() + ":" + getMinutes() + ":" + getSeconds());

String result = s.toString();

return(result + getDayTime());

}

}

TimeOfDayTester.java

public class TimeOfDayTester

{

public static void main(String[] args)

{

TimeOfDay test1 = new TimeOfDay(11,59,30,"p");

TimeOfDay test2 = new TimeOfDay(12,59,30,"a");

TimeOfDay test3 = new TimeOfDay(10,59,30,"p");

System.out.println("Test 1: " + test1.toString());

System.out.println("Test 2: " + test2.toString());

System.out.println("Test 3: " + test3.toString());

System.out.println("Seconds between Test 1 and Test 2: " + test1.secondsFrom(test2));

System.out.println("Seconds between Test 1 and Test 3: " + test1.secondsFrom(test3));

System.out.println("Seconds between Test 2 and Test 3: " + test2.secondsFrom(test3));

System.out.println("Adding 30 seconds to Test 1 gives: " + test1.addSeconds(30).toString());

}

}

1. (10 points)

Cohesion: this class violates the PoLS because one would think that constructor would throw some kind of error since height and width should be non-negative, but the constructors allow negative input which results in the undefined behavior of the object methods.

Completeness: this class is incomplete because there are no mutator methods for width and height while there are accessor methods for them. You cannot set their values individually in this class since there are only mutator methods like setSize() .

Convenience: This class is not convenient because of the lack of individual mutator methods for setting the height and width, and its lack of clarity makes it difficult to understand how to even use it.

Clarity: There is no clarity in this class because the constructor Dimension(Dimension d) creates an instance of Dimension with the same width and height, but this is not clear unless explained. If there is a constructor such as Dimension(int w, int h), then there is no need for the constructor mentioned previously because this constructor handles this situation. The equals() method is unclear because it doesn’t describe what properties of the objects it is comparing in order to determine if they are equal or not.

Consistency: this class is also not consistent because the prefixes are used inconsistently as in hashCode() and getHeight(). It is also inconsistent because the fields: height and width are integers, yet the accessor methods return the height and width as a double. In addition, there are two setSize() methods in which one takes in integers and the other takes in doubles which is unnecessary and confusing if the weight and height are supposed to be stored as integers.

**General Notes:**

When you are ready, submit your classes, your Javadoc output, and your testing; the classes and the testing have to also be in hard copy.\

Put all the comments about testing in a Tester class. Defend the test data strategy in the comment, also. Document each constructor and method. Clear programming style will account for a substantial portion of your grade for the programming part of this assignment.

**Checklist:**

For programming: Source code, submitted as part of the tarball, by the electronic submission deadline. All testing runs (cut and pasted from console, and/or captured screenshots), all Javadoc in html form. Name and UNI visible in all files: source, test runs, Javadoc. You can provide a ReadMe file if you believe it will help the TAs understand the overview of your system design and testing, but it is much better if any such documentation is integral to the Tester class instead.

# tar -cvzf myUNI\_HW2.tar.gz whateverMyDirectoryForHW1Is