Nicholas Liu (nl2523)

Assignment 2 - Theory

1.

a. The three axioms that partial order must obey are:

* Reflexive: Is A == A? (True)
* Antisymmetric: If A > B and B > A then triangle A is the same as triangle B. (True)
* Transitive: If there were another triangle C where B contains C, then A > B and B > C. Therefore A > C (True).
* Since all these points are true, this is a partial order.

b. Assume we have Triangle A has an area of 3 and a perimeter of 7.9 and Triangle B has an area of 27 and a perimeter of 23.69. We can compare the area with the area and the perimeter with the perimeter but for it to be a total order we must looks at the triangle and compare area 3 with the perimeter 23.69. However, this comparison is meaningless as it is comparing area with perimeter and thus is not a total order.

c. To find a total order you lexographically order the coordinates until you find one that is not equal. Then you can order based on which character comes first. However, this method is incredibly inefficient.

2.

import java.util.Calendar;

import java.util.GregorianCalendar;

public class Ass2\_2 {

public static void main(String args[]) {

final int MILLI\_SEC = 1000, SEC\_MIN = 60, MIN\_HOUR = 60, HOUR\_DAY = 24;

final long MILLI\_DAY = MILLI\_SEC \* SEC\_MIN \* MIN\_HOUR \* HOUR\_DAY;

long timeOfFu, timeOfCurrent;

int daysSince;

GregorianCalendar calendar = new GregorianCalendar();

timeOfCurrent = calendar.getTimeInMillis();

calendar.set(1997, Calendar.*OCTOBER*, 1);

timeOfFu = calendar.getTimeInMillis();

daysSince = (int) Math.*floor*((timeOfCurrent - timeOfFu) / MILLI\_DAY);

System.*out*.println("There have been " + daysSince + " days since Fu donated on October 1, 1997.");

}

}

3.

public class TimeOfDay\_Tester {

public static void main(String args[]) {

TimeOfDay today = new TimeOfDay(8, 42, 102);

System.*out*.println("Hours: " + today.getHours() + " Minutes: " + today.getMinutes() + " Seconds: " + today.getSeconds());

System.*out*.println("There are " + today.getSecondsRemaining() + " seconds until midnight");

TimeOfDay tomorrow = today.addSeconds(100);

System.*out*.println("Between: " + today.secondsFrom(tomorrow));

TimeOfDay error = new TimeOfDay(23, 59, 62);

}

}

public class TimeOfDay {

public int hours, minutes, seconds;

public final int SEC\_MIN = 60, MIN\_HOUR = 60; //Constants for conversions

public final int MIDNIGHT\_SEC = 59, MIDNIGHT\_MIN = 59, MIDNIGHT\_HOUR = 23;

//Default constructor that sets time to 0

public TimeOfDay() {

hours = 0;

minutes = 0;

seconds = 0;

}

//Constructor that takes in user input

public TimeOfDay(int hours, int minutes, int seconds) {

this.setHours(hours);

this.setMinutes(minutes);

this.setSeconds(seconds);

}

public int getHours() {

return hours;

}

public void setHours(int hours) {

try {

if (hours >= 24)

throw new IllegalArgumentException();

} catch (IllegalArgumentException e) {

System.*out*.println("Invalid: Hours must be < 24");

}

this.hours = hours;

}

public int getMinutes() {

return minutes;

}

public void setMinutes(int minutes) {

if (minutes > 59)

setHours(hours + (minutes / 60));

this.minutes = minutes % 60;

}

public int getSeconds() {

return seconds;

}

public void setSeconds(int seconds) {

if (seconds > 59)

setMinutes(minutes + (seconds / 60));

this.seconds = seconds % 60;

}

public int getSecondsRemaining() {

TimeOfDay midnight = new TimeOfDay(MIDNIGHT\_HOUR, MIDNIGHT\_MIN, MIDNIGHT\_SEC);

return secondsFrom(midnight) + 1; //+1 because we're looking for midnight not 23:59:59

}

public int secondsFrom(TimeOfDay other) {

int otherHours = other.getHours(), otherMinutes = other.getMinutes(), otherSeconds = other.getSeconds();

int thisHours = this.getHours(), thisMinutes = this.getMinutes(), thisSeconds = this.getSeconds();

int totalOtherSeconds = (otherHours \* MIN\_HOUR \* SEC\_MIN) + (otherMinutes \* SEC\_MIN) + otherSeconds;

int totalThisSeconds = (thisHours \* MIN\_HOUR \* SEC\_MIN) + (thisMinutes \* SEC\_MIN) + thisSeconds;

int totalSeconds = Math.*abs*(totalThisSeconds - totalOtherSeconds);

return totalSeconds;

}

public TimeOfDay addSeconds(int addedSeconds) {

TimeOfDay returnValue = new TimeOfDay(hours, minutes, seconds + addedSeconds);

return returnValue;

}

}

4.

Cohesion: Using the Sesame Street Test, there does not seem to be any names that would appear to be an outlier. All of them start with a verb and then if there is a second word, it is a noun

Completeness: For the most part, the LinkedList class is complete. For example, there is an addFirst method that adds an element to the first position. Likewise, there is a complementary removeFirst method that removes the first and a getFirst that gets the first element. However, there is not explicit method to set the first index but there is a method that the user can use to set the value of any position so in that sense, the “First” methods are fairly complete. However, there are also some more blatant holes to this class’ completeness. There is a method called “descendingIterator” that returns an iterator in reverse of sequential order. However, there is not complementary method called ascendingIterator and therefore it fails the complementary test.

Convenience: The LinkedList is pretty convenient. Whenever anyone needs to retrieve an item, it will return whatever kind of element is in the list.

Clarity: Most of the class is following the Principle of Least Surprise as nothing is jumping out at me. I also feel like I can follow through with the documentation.

Consistency: Nothing in LinkedList is deprecated in any version of Java. Therefore, the class is very consistent.