**Lab 16 20-11-2017**

**Object Oriented Concepts and Programming**

**Learning Objectives**

**Topics:** interfaces and their implementation

Teacher’s perspective

1. Understanding the purpose of interfaces
2. Comprehend the limitations of defining interfaces
3. Demonstrate the separation of interface from implementation
4. The students will be able to

* Create and implement interfaces
* Provide implementation of interfaces
* Handle polymorphic interface variables to invoke polymorphic calls on unrelated objects

|  |
| --- |
| **Lab Walkthrough/Demo** |

**Lab Demo 15.1.**

The Ticker interface

public interface Ticker

{

void Tick();

}

**The Date class**

/\*

\* The Date class models a calendar date with day, month and year.

\* This class does not perform input validation for day, month and year.

\*/

public class Date implements Ticker {

// The private instance variables

private int year, month, day;

// The constructors

public Date(int year, int month, int day) {

// No input validation

this.year = year;

this.month = month;

this.day = day;

}

// The public getters/setters for the private variables

public int getYear() {

return this.year;

}

public int getMonth() {

return this.month;

}

public int getDay() {

return this.day;

}

public void setYear(int year) {

this.year = year; // No input validation

}

public void setMonth(int month) {

this.month = month; // No input validation

}

public void setDay(int day) {

this.day = day; // No input validation

}

// Return "MM/DD/YYYY" with leading zero for MM and DD.

public String toString() {

// Use built-in function String.format() to form a formatted String

return String.format("%02d/%02d/%4d", month, day, year);

// Specifier "0" to print leading zeros, if available.

}

// Set year, month and day - No input validation

public void setDate(int year, int month, int day) {

this.year = year;

this.month = month;

this.day = day;

}

**// Increment this instance by one day**

**public void Tick() {**

**++day;**

**if (day >= 30) { // ignore days per month**

**day = 1;**

**++month;**

**if (month >= 12) {**

**month = 1;**

**++year;**

**}**

**}**

**}**

}

**The Time class**

/\*

\* The Time class models a time instance with second, minute and hour.

\* This class does not perform input validation for second, minute and hour.

\*/

public class Time implements Ticker{

// The private instance variables

private int second, minute, hour;

// The constructors (overloaded)

public Time(int second, int minute, int hour) {

// No input validation

this.second = second;

this.minute = minute;

this.hour = hour;

}

public Time() { // the default constructor

this.second = 0;

this.minute = 0;

this.hour = 0;

}

// The public getters/setters for the private variables.

public int getSecond() {

return this.second;

}

public int getMinute() {

return this.minute;

}

public int getHour() {

return this.hour;

}

public void setSecond(int second) {

this.second = second; // No input validation

}

public void setMinute(int minute) {

this.minute = minute; // No input validation

}

public void setHour(int hour) {

this.hour = hour; // No input validation

}

// Return "hh:mm:ss" with leading zeros.

public String toString() {

// Use built-in function String.format() to form a formatted String

return String.format("%02d:%02d:%02d", hour, minute, second);

// Specifier "0" to print leading zeros, if available.

}

// Set second, minute and hour

public void setTime(int second, int minute, int hour) {

// No input validation

this.second = second;

this.minute = minute;

this.hour = hour;

}

**// Increment this instance by one second**

**public void Tick() {**

**++second;**

**if (second >= 60) {**

**second = 0;**

**++minute;**

**if (minute >= 60) {**

**minute = 0;**

**++hour;**

**if (hour >= 24) {**

**hour = 0;**

**}**

**}**

**}**

**}**

}

**The TickerTest class**

**What will be the output?**

/\*

\* A Test Driver for the Date class.

\*/

public class TickerTest {

public static void main(String[] args) {

// Test constructor and toString()

Date d1 = new Date(2016, 12, 30);

System.out.println(d1); // toString()

Time t1 = new Time(1, 2, 3);

System.out.println(t1); // toString()

// Test setTime()

t1.setTime(59, 59, 23);

System.out.println(t1); // toString()

// Test polymorphic reference of Ticker interface

Ticker dateTimeTicker;

dateTimeTicker = t1;

dateTimeTicker.Tick();

System.out.println(t1); // toString()

dateTimeTicker = d1;

dateTimeTicker.Tick();

System.out.println(d1); // toString()

}

}

**OUTPUT:**

12/30/2016

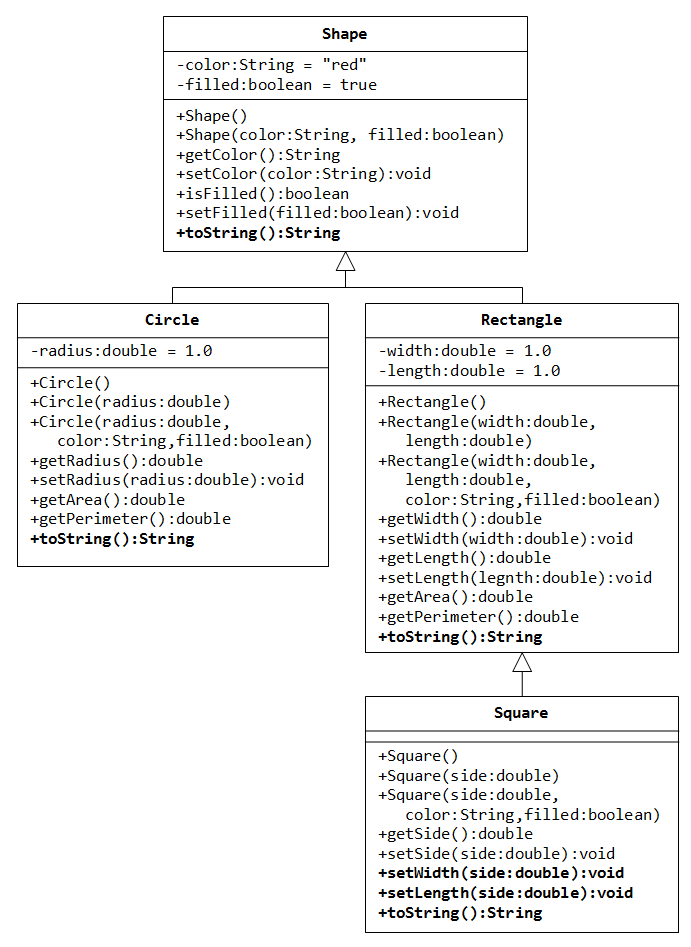
03:02:01

23:59:59

00:00:00

01/01/2017

|  |
| --- |
| **Lab Task** |

**Lab Task 1**

**Problem Statement**

Download the classes (Shape and Rectangle) from Google classroom.

Add all the classes in an eclipse package.

1. The StatsComputable interface: Create an interface named **StatsComputable** containing 2 methods called **computeStats() and reset()**.
   * void computeStats();
     + It computes different statistics of the object, which implements the StatsComputable interface.
   * void reset();
     + The reset method resets all those fields to 0, which are involved in computing the stats (i.e. area)
   * The interface also contains a final string field called units initialized to “KM^2”, which represent the units of area.
2. The Shapes Hierarchy: The **Shape** class should now implement the StatsComputable interface. However, the concrete implementation of computeStats() and reset() methods should be provided in the **Rectangle** class. So, make the following changes to the **Rectangle** class.
   * Add a couple of fields named rectArea and rectPerimeter to the Rectangle class. These fields will store the stats computed by the computeStats method, which should be overridden in the Rectangle class.
   * Override the reset method, which should reset rectArea and rectPerimeter to 0.
   * Implement the getter methods getRectArea() and getRectPerimeter()**.** 
     + **[OPTIONAL]** Implement the getter methods for rectArea and rectPerimeter, so that they should throw an exception of type **InvalidAttributeValueException**, if rectArea or rectPerimeter is 0, otherwise, they should return rectArea or rectPerimeter value, respectively.
   * Implement the toString method returning a string containing the rectangle width, height, and area/perimeter in the format of your own choice.
3. The Country Class: Create a class called country, which has following fields:
   * + countryName (string)
     + provincesAreas (an array of integers)
     + statesAreas (an arrayList of integers)
     + countryArea (integer)
   * This class implements the computeStats() method of the StatsComputable interface, which computes the area of country by adding together the areas of all the provinces and states and saves it to the countryArea field.
   * This class also implements the reset() method of StatsComputable interface, which reset countryArea to 0.
   * Implement the getter method getCountryArea() for countryArea **.**
     + **[OPTIONAL]** Implement the getter method for countryArea, so that it throws an exception of type **InvalidAttributeValueException** if countryArea is 0, otherwise, it should return countryArea value.
   * Implement the toString method returning a string containing the country name and area along with units field from the statsComputable interface in the format of your own choice.
4. The Test Class:Create a test class named TestStatsComputable,
   * Inside the main method, create 2 objects of Rectangle and Country classes, respectively.
     + Call the printStatsComputable method (see below for details) for each of the created objects to print its area and reset its StatsComputable.
     + [OPTIONAL]: Call the getter methods getRectArea() and getCountryArea() methods of both objects. These methods should throw an exception of type **InvalidAttributeValueException** (as the statistics were reset to 0 by printStatsComputable method)**,** which should be caught in your code and appropriate message is shown.
   * Create a static method named Void **printStatsComputable**(StatsComputable i); This method calls the computeStats method to print the area of the passed object. Then, the method should display the passed object using its toString method. Note that these methods will be called polymorphically. Before exiting the method, it also calls the reset method to reset the object statistics.