**Online Lab 8**

**Object Oriented Programming**

**Learning Objectives**

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** |

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

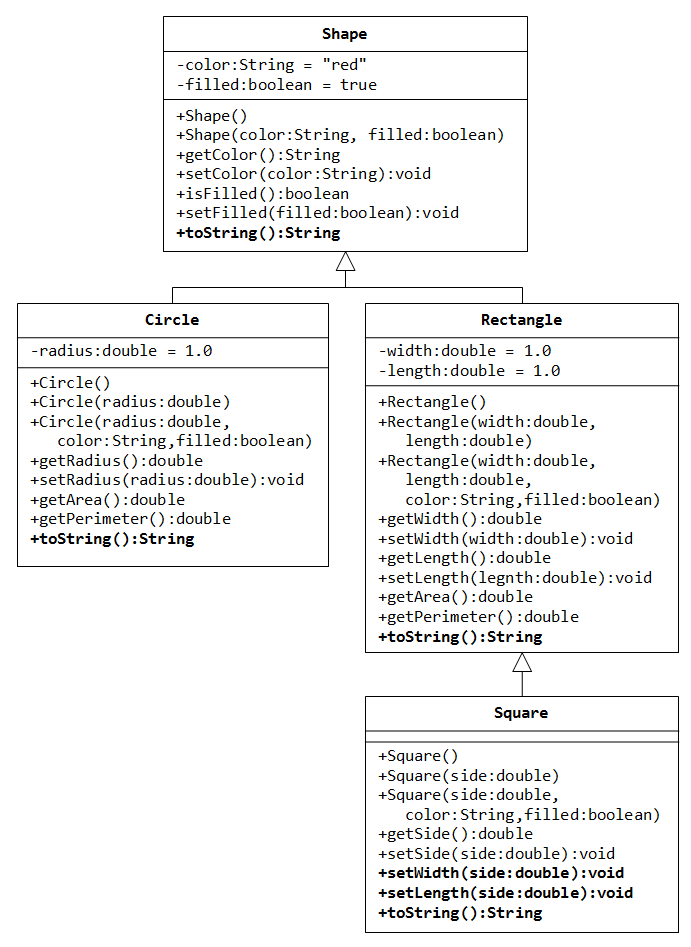
03:02:01

23:59:59

00:00:00

01/01/2017

|  |
| --- |
| **Home Task** |



Use the above class hierarchy and modify the code as follows:

Add all the classes in an eclipse package.

1. The Stats interface: Create an interface named **Stats** containing 2 methods called **computeArea() and reset()**.
   * void computeArea();
     + The computeArea method returns the area
   * 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 **string field** called units, which represent the units of area (e.g. square kilometer – “sq.km” or square meter – “sq.m” etc.)
2. The Shapes Hierarchy: The **Shape** class should now implement the Stats interface. But, the concrete implmentation of computeArea() and reset() methods should be provided in the **Rectangle** class.
   * The computeArea method should print the area of rectangle along with units.
   * The reset method should reset width and height to 0.
3. The Country Class: Create a class called country, which has following fields:
   * name
   * provincesAreas (an array of integers)
   * statesAreas (an arrayList of integers)
   * This class implements the computeArea() method of Stats interface. The area of country is computed by adding together all the areas of provinces and states.
   * This class also implments the reset() method of Stats interface. Reset all the values of provincesAreas and statesAreas to 0.
4. The Test Class:Create a test class named TestStats,
   * Create a static method named
     + Void printStats(interface i);
     + Which calls the computeArea method to print the area of passed object. Before exiting the method, also call the reset method to reset the object statistics.
   * Inside the main method, create 3 objects of Rectangle, Square and Country classes, polymorphically
     + Call the printStats method for each of the created objects to print its area and reset its stats using an enhanced for.