National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Department of Computing

CS 212: Object Oriented Programming

Class: BESE-7AB

Lab 04: Classes & Objects II

Date: March 17th, 2017

Instructor:

Dr. Muhammad Muneeb Ullah

Learning Objectives

The learning objective of this lab is to understand and practice the concepts of constructors, constructor overloading, get and set methods, this reference and overloaded methods.

Activity #1

Complete the following Time Class and provide the missing constructor definitions. For each constructor, you need to use the **setTime** method for setting-up the parameters that are passed to it. In this way, you can ensure that all parameters are valid. Also, see what happens if you pass an invalid parameter to the constructor, while creating an object of Time class (as done for t4 object).

```
class Time
{
   private int hour;   // 0 - 23
   private int minute; // 0 - 59
   private int second; // 0 - 59

   // TODO: Add constructors definitions here

   // set a new time value using universal time; throw an
   // exception if the hour, minute or second is invalid
   public void setTime( int h, int m, int s )
   {
      // validate hour, minute and second
      if ( ( h >= 0 && h < 24 ) && ( m >= 0 && m < 60 ) && ( s >= 0 && s < 60 ) )
      {
         hour = h;
         minute = m;
         second = s;
      } // end if
      else
         throw new IllegalArgumentException("hour, minute and/or second was out
of range" );
   } // end method setTime

   // convert to String in universal-time format (HH:MM:SS)
   public String toUniversalString()
   {
      return String.format( "%02d:%02d:%02d", hour, minute, second );
   } // end method toUniversalString
```

```
   // convert to String in standard-time format (H:MM:SS AM or PM)
   public String toString()
   {
      return String.format( "%d:%02d:%02d %s",
                        ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 ),
                        minute, second, ( hour < 12 ? "AM" : "PM" ) );
   } // end method toString
} // end class Time

// This class tests the Time Class by creating different objects.
public class TimeTest{
   public static void main(String [] args){
      Time t0 = new Time();          // Set Time to 00:00:00
      Time t1 = new Time(11);        // Set Time to 11:00:00
      Time t2 = new Time(12, 40);    // Set Time to 12:40:00
      Time t3 = new Time(23, 40, 55); // Set Time to 23:40:55
      Time t4 = new Time(23, 40, 65); // Set Time to 23:40:65

      // Print All Times in Universal Format
      System.out.println(t0.toUniversalString());
      System.out.println(t1.toUniversalString());
      System.out.println(t2.toUniversalString());
      System.out.println(t3.toUniversalString());

      // Print All Times in Standard Format
      System.out.println(t0);
      System.out.println(t1);
      System.out.println(t2);
      System.out.println(t3);
   }
}
```

Activity #2

What will be the output of the following program? Execute and confirm your understanding.

```
// Demonstrate method overloading.
class Overload {
   void test() {
      System.out.println("No parameters");
   }
   // Overload test for one integer parameter.
   void test(int a) {
      System.out.println("a: " + a);
   }
   // Overload test for two integer parameters.
   void test(int a, int b) {
      System.out.println("a and b: " + a + " " + b);
   }
   // Overload test for a double parameter
   double test(double a) {
      System.out.println("double a: " + a);
      return a*a;
   }
}

public class OverloadTest {
   public static void main(String args[]) {
      Overload ol = new Overload();
      double result;

      // call all versions of test()
      ol.test();
      ol.test(10);
      ol.test(10, 20);
      result = ol.test(123.2);

      System.out.println("Result of ol.test(123.2): " + result);
   }
}
```

Activity #3

There are some issues with the following definition of the "Overload" Class. Find them out?

```
// Demonstrate method overloading.
class Overload {
   void Overload(){
      System.out.println("Constructor Called");
   }

   void test() {
      System.out.println("No parameters");
   }

   // Overload test for one integer parameter.
   void test(int a) {
      System.out.println("a: " + a);
   }

   // Overload test for one integer parameter.
   int test(int a) {
      System.out.println("a: " + a);
      return 0;
   }

   // Overload test for two integer parameters.
   void test(int a, int b) {
      System.out.println("a and b: " + a + " " + b);
   }

   // Overload test for two integer parameters.
   void test(int b, int a) {
      System.out.println("a and b: " + a + " " + b);
   }

   // Overload test for a double parameter
   double test(double a) {
      System.out.println("double a: " + a);
      return a*a;
   }
}
```

Task #1:

An integer number is said to be a perfect number if its factors, including 1 (but not the number itself), sum to the number. For example, 6 is a perfect number, because 6 = 1 + 2 + 3. Write a method isPerfect that determines if parameter number is a perfect number. Use this method in an application that displays all the perfect numbers between 1 and 1000. Display the factors of each perfect number to confirm that the number is indeed perfect. Challenge the computing power of your computer by testing numbers much larger than 1000. Display the results.

Task #2:

Create class SavingsAccount . Use a static variable annualInterestRate to store the annual interest rate for all account holders. Each object of the class contains a private instance variable savingsBalance indicating the amount the saver currently has on deposit. Provide method calculateMonthlyInterest to calculate the monthly interest by multiplying the savingsBalance by annualInterestRate divided by 12—this interest should be added to savingsBalance.

Provide a static method modifyInterestRate that sets the annualInterestRate to a new value. Write a program to test class SavingsAccount. Instantiate two savingsAccount objects, saver1 and saver2 , with balances of $2000.00 and $3000.00, respectively. Set annualInterestRate to 4%, then calculate the monthly interest for each of 12 months and print the new balances for both savers. Next, set the annualInterestRate to 5%, calculate the next month's interest and print the new balances for both savers.

Task #3:

Modify class Time2 of (as given below) to include a tick method that increments the time stored in a Time2 object by one second. Provide method incrementMinute to increment the minute by one and method incrementHour to increment the hour by one.

Write a program that tests the tick method, the incrementMinute method and the incrementHour method to ensure that they work correctly. Be sure to test the following cases:

a) incrementing into the next minute,
b) incrementing into the next hour and
c) incrementing into the next day (i.e., 11:59:59 PM to 12:00:00 AM).

```
public class Time2
{
   private int hour;   // 0 - 23
   private int minute; // 0 - 59
   private int second; // 0 - 59
```

```
// Time2 no-argument constructor:
// initializes each instance variable to zero
public Time2()
{
   this( 0, 0, 0 ); // invoke Time2 constructor with three arguments
} // end Time2 no-argument constructor

// Time2 constructor: hour supplied, minute and second defaulted to 0
public Time2( int h )
{
   this( h, 0, 0 ); // invoke Time2 constructor with three arguments
} // end Time2 one-argument constructor

// Time2 constructor: hour and minute supplied, second defaulted to 0
public Time2( int h, int m )
{
   this( h, m, 0 ); // invoke Time2 constructor with three arguments
} // end Time2 two-argument constructor

// Time2 constructor: hour, minute and second supplied
public Time2( int h, int m, int s )
{
   setTime( h, m, s ); // invoke setTime to validate time
} // end Time2 three-argument constructor

// Time2 constructor: another Time2 object supplied
public Time2( Time2 time )
{
   // invoke Time2 three-argument constructor
   this( time.getHour(), time.getMinute(), time.getSecond() );
} // end Time2 constructor with a Time2 object argument

// Set Methods
// set a new time value using universal time;
// validate the data
public void setTime( int h, int m, int s )
{
   setHour( h ); // set the hour
   setMinute( m ); // set the minute
   setSecond( s ); // set the second
} // end method setTime
```

```
// validate and set hour
public void setHour( int h )
{
   if ( h >= 0 && h < 24 )
      hour = h;
   else
      throw new IllegalArgumentException( "hour must be 0-23" );
} // end method setHour

// validate and set minute
public void setMinute( int m )
{
   if ( m >= 0 && m < 60 )
      minute = m;
   else
      throw new IllegalArgumentException( "minute must be 0-59" );
} // end method setMinute

// validate and set second
public void setSecond( int s )
{
   if ( s >= 0 && s < 60 )
      second = ( ( s >= 0 && s < 60 ) ? s : 0 );
   else
      throw new IllegalArgumentException( "second must be 0-59" );
} // end method setSecond

// Get Methods
public int getHour(){
   return hour;
}

public int getMinute(){
   return minute;
}

public int getSecond(){
   return second;
}
```

```
   // convert to String in universal-time format (HH:MM:SS)
   public String toUniversalString()
   {
      return String.format("%02d:%02d:%02d", getHour(), getMinute(),
getSecond() );
   }

   // convert to String in standard-time format (H:MM:SS AM or PM)
   public String toString()
   {
      return String.format( "%d:%02d:%02d %s",
                  ( (getHour() == 0 || getHour() == 12) ? 12 : getHour() % 12 ),
                  getMinute(), getSecond(), ( getHour() < 12 ? "AM" : "PM" ) );
   } // end method toString
} // end class Time2
```

Hand in

Hand in the source code from this lab at the appropriate location on the LMS system. You should hand in a single compressed/archived file named Lab_4_<Your CMS_ID. Your_NAME >.zip (without angle brackets) that contains ONLY the following files.

1) All completed java source files representing the work accomplished for this lab. The Java source files should contain author in the comments at the top.

2) A plain text file named **README.TXT** that includes a) author information at the beginning, b) a brief explanation of the lab, and c) any comments, or suggestions.

To Receive Credit

1. By showing up on time for lab, working on the lab solution, and staying to the end of the class period, only then you can receive full credit for the lab assignment.

2. Comment your program heavily. Intelligent comments and a clean, readable formatting of your code account for 20% of your grade.

3. The lab time is not intended as free time for working on your programming/other assignments. Only if you have completely solved the lab assignment, including all challenges, and have had your work checked off for completeness by your TA/Lab Engineer should you begin the programming/other assignments.