**CIS 163 Project 1**

**A SimpleDate program**

**Due Date**

* At the beginning of the lab; see the schedule, last page of the syllabus

**Before Starting the Project**

* Review Chapter 1 – 5 of the CIS163 book
* Read this entire project description before starting

**Learning Objectives**

After completing this project you should be able to:

* Have a good working knowledge of the topics covered in CIS162
* Create classes with associated methods
* Use complex if statements
* Read and write data from external text files
* Use static methods and properties available in the Java library
* Use the internet and API to create a Timer object

**You must complete each step fully before proceeding on. No credit is given to any given step unless the previous steps have been completed and are functioning!**

**Before you turn in your work: use the** [**Java Style Guide**](http://www.cis.gvsu.edu/studentsupport/javaguide) **to document your project. (10 pts)**

**Step 1: Create an Eclipse project named “SimpleDatePrj”**

* Create a project named: project1 (right click on “SimpleDatePrj”)
* Create a package named: package1 (right click on “Projects” and select package)
* Create a class named: SimpleDate (right click on “package1” and select new/class)
  + The properties and methods for this class are in step 2.
* Create a JUnit Test Case named: TestSimpleDate (right click on “SimpleDate”and select new/JUnit Test Case)
  + Log on to BB and cut and paste the starting files found in the project 1 folder under Course Documents.

**Step 2: Implement the following methods for the class “SimpleDate”**

Implement the following methods and properties in SimpleDate class. For properties, you will need three instance variables: day (integer), month (integer), year (integer). For methods, you will need to implement the following (include any setters or getters as needed). Unless otherwise stated, you can assume the input has no errors (i.e., a valid set of numbers) for this step. Later steps in the project will remove this assumption.

* public SimpleDate() Default constructor that sets the SimpleDate to zero
* public SimpleDate(int day, int month, int year) A constructor that initializes the instance variables with the provided values
* public SimpleDate (SimpleDate other) A constructor that initializes the instance variables with the other SimpleDate parameter
* public SimpleDate(String startDate) A constructor that accepts a string as a parameter with the following format: “9/21/2013” where 21 indicates day, 9 indicates month, and 2013 indicates year. You can assume the input has no errors (i.e., a valid set of numbers) contained within and the year is a 4 digit number
* public boolean equals(Object other) A method that returns true if “this” SimpleDate object is exactly the same as the other object (Note: you must cast the other object as a SimpleDate object)
* public static boolean equals(SimpleDate s1, SimpleDate s2) A method that returns true if SimpleDate s1 is exactly the same as SimpleDate s2 object.
* public int compareTo(SimpleDate other) A method that returns 1 if “this” SimpleDate object is greater than the other SimpleDate object; returns -1 if the “this” SimpleDate object is less than the other SimpleDate; returns 0 if the “this” SimpleDate object is equal to the other SimpleDate object
  + For example: “12/12/2012” is less than “1/1/2013”
* Public boolean isLeapYear() A method that returns true if “this” year is a leap year
* Public static boolean isLeapYear(int year) A method that returns true if year is a leap year
* Public int ordinalDate() A method the returns the number of days at the beginning of the “this” year.
  + For example: “2/10/2013” returns 41. (e.g., 31 + 10)
  + Be care about leap year
* public void increment() A method that increments the “this” SimpleDate by 1 day
  + Note: 12/31/2012 incremented would result in: 1/1/2013
* public String toString() Method that returns a string that represents a SimpleDate with the following format: “Day Month Year”. For example: “21 March 2013”. Display the day as is with leading “0” if day is less than 10. For months, use the English words that specifics the month, e.g. January, February, etc. Finally, display year with 4 digits.
* public static int getNumberOfSimpleDates() Returns the number of SimpleDate objects created by your program. Hint: use a static variable and increment that variable for each instantiation.

**Step 3: Software Testing: Using a JUnit named “SimpleDateTest”;** software developers must plan from the beginning that their solution is correct.

* Within this file you will see comments on where to place the JUnit test cases.

**Step 4: Create the following additional methods in the SimpleDate class:**

* public void save(String fileName)A method that saves the “this” SimpleDate to a file; use the parameter filename for the name of the file
* public void load(String fileName)A method that loads the “this” SimpleDate object from a file; use the parameter filename for the name of the file
* **A change in step 2**. Allow for an error in the input arguments for **all** constructors and methods, and throw a IllegalArgumentException exception if an error occurs. Also, there is a minimal date allowed for all methods: 1/1/1753. Example one, “13/13/2013” is not a valid input string for a constructor in step 2 and an IllegalArgumentException is thrown. Example two, “1/1/1700” is not a valid input date.
  + Sample code snippet that may help:

if (month > 12)

throw new IllegalArgumentException();

**Following will help you with reading and writing to a file:**

The data file is shown directly below and contains only one line of data. Your book (and google) shows the use of the Scanner Class.

ANCHORAGE 256000

**Here is the code that would read the above file:**

**public** **void** sampleReadData(){

String city;

**int** population;

**try**{

// open the data file

Scanner fileReader = **new** Scanner(**new** File("/testit"));

// read one String in of data and an int

city = fileReader.next();

population = fileReader.nextInt();

System.*out*.println (city + " " + population);

}

// could not find file

**catch**(FileNotFoundException error) {

System.*out*.println("File not found ");

}

// problem reading the file

**catch**(IOException error){

System.*out*.println("Oops! Something went wrong.");

}

}

**Here is the code that would write the above file:**

**public** **void** sampleWriteData () {

PrintWriter out = **null**;

**try** {

out = **new** PrintWriter(**new** BufferedWriter(**new** FileWriter("/testit")));

}

**catch** (IOException e) {

e.printStackTrace();

}

String s = "ANCHORAGE";

out.println(s + " " + "256000");

out.close();

}

**Step 5: Challenge Requirement (7 pts)**

* The following should only be attempted after all of the other requirements have been completed.

public SimpleDate daysFromNow(int n) throws IllegalArgumentException is a method that returns a new SimpleDate object representing the date “n” days from now. If “n” is negative, then return the date “n” days in the past. Throw an IllegalArgumentException for any error that occurs, for example, if the resulting date is before the minimum allowable date. In the above steps, you created an ‘increment’ method, that method maybe useful. (Perhaps a decrement method maybe useful).

**Step 6: Challenge Requirement (8 pts)**

* The following should only be attempted after all of the other requirements have been completed.

Public int daysSince (Simple Date other) Create a method named that returns the number of days that have elapsed from “this” simpleDate to “other” simpleDate. If “other” follows “this” date, then the value returned is negative. Throw an IllegalArgumentException for any input error.

--------------------------- YOUR’RE DONE ☺ -------------------------------

**Some additional grading criteria**

There is a 70% penalty on programming projects if your solution does not compile.

* Stapled cover page with your name and signed pledge. (-5 pts if missing)

**Late Policy**

Projects are due at the START of the class period, important, the first 24 hours late is -15 pts

* Each subsequent weekday is an additional -10 pts

**Turn In**

A professional document is stapled with an attractive cover page.

* Cover page - Your project must have a cover page that includes your name, a title, an interesting graphic or photograph related to the project topic and the following signed pledge: "I pledge that this work is entirely mine, and mine alone (except for any code provided by my instructor). " You are responsible for understanding and adhering to the [School of CIS Guidelines for Academic Honesty](http://www.cis.gvsu.edu/Academics/Honesty/).

**Project 1: “SimpleDate” Program Rubric.**

|  |  |
| --- | --- |
| Student Name |  |
| Date Submitted, Days Late, Late Penalty |  |

|  |  |  |
| --- | --- | --- |
| **Graded Item** | **Points** | **Comments and Points Secured** |
| Javadoc Comments and Coding Style/Technique  (<http://www.cis.gvsu.edu/studentsupport/javaguide>)  Here are some examples but not limited to:   * Code Indentation (auto format source code in IDE) * Naming Conventions (see Java style guide) * Proper access modifiers for fields and methods * Use of helper (private) methods * Using good variable names * Header/class comments * Every method uses @param and @return * Every method uses a /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* separator * Overall layout, readability, No text wrap * Using /\*\* … / for each Instance variable * Has many inner “inner” comments | 10 |  |
| **Base Functionality**   * SimpleDate() * SimpleDate(int, int, int) * SimpleDate(SimpleDate other) * SimpleDate(String date) * getNumberOfSimpleDates() * equals(Object other) * equals(SimpleDate s1, SimpleDate s2) * compareTo(SimpleDate other) * isLeapYear() * isLeapYear(int year) * ordinalDate() * increment() * toString() | 50 |  |
| **JUnit test** | 10 |  |
| **Additional Methods**   * save(String fileName) * load(String fileName) * daysFromNow(int n) * daysSince(SimpleDate other) | 5  5  5  5 |  |
| **Parameter/Input Validation** | 10 |  |
| **Total** | **100** |  |

**Additional Comments:**