# **CIS 162 Project 1**

# **GVdate**

### **Due Date**

week 7 – See due date on Blackboard for your specific section.

Suggested guidelines for the amount of time required for each phase:

* Phase 1 – One – two days
* Phase 2 – One week
* Phase 3 – One week

### **Before Starting the Project**

* Read chapter 5, chapter 6 and chapter 7 (sections 7.1 – 7.4)
* Read this entire project description before starting

### **Learning Objectives**

After completing this project, you should be able to:

* *write* a complete Java class
* *write* methods to meet specific requirements
* *write* conditional statements with Boolean expressions
* *write* simple loops
* *explain* the differences between local variables, instance variables (class fields) and method parameters

### **Note**

Do not use the existing Java Date class in any way.

### **For Your Reference**

Recall that there are 365 days in a year (except a leap year that has 366 days). The months and number of days for each month are provided:

January (31), February (28 or 29 – depending on whether or not is a leap year), March (31), April (30), May (31), June (30), July (31), August (31), September (30), October (31), November (30), December (31)

Your code should follow the [CIS 162 Programming Style Guide](https://www.cis.gvsu.edu/java-coding-style-guide/). (10 pts)

# Phase 1 (20 pts)

### **Requirements**

Create a class called GVdate to keep track of a calendar date including month, day and year. You can do simple things like checking if it is your birthday, advancing to the next day, checking if a given date is valid and checking if it is a leap year.

**Important Note:**

**We are providing JUnit classes to test your project. For these classes to compile, exact spelling is required for the name of the GVdate class. Do not change in any way the headers of the methods of the GVdate class.**

### Class Fields/Instance Variables

* Provide appropriate names and data types for each of the private instance variables:
  + the month, day and year (int)
  + two **final integers** that represent YOUR birthday (month and day)

### Constructors

* public GVdate( )– this is the default constructor; it assigns the following initial values to the instance variables:
  + month: 10
  + day: 12
  + year: 2020
* public GVdate (int m, int d, int y) – set the instance variables to the provided input parameters. For now, assume that the input parameters are correct.

### Accessor Methods

* public int getMonth( ) – return the month.
* public int getDay( ) – return the day.
* public int getYear( ) – return the year
* public String toString() – return a formatted date string (for example "10/12/2020").
* public boolean isMyBirthday () – return true if the date is YOUR birthday. Otherwise return false.

### Mutator Methods

* public void setMonth(int m)- set the month to the input parameter. For now, assume the value for the month will not create an invalid date.
* public void setDay(int d)- set the day to the input parameter. For now, assume the value for the day will not create an invalid date.
* public void setYear(int y)- set the year to the input parameter. For now, assume the value for the year will not create an invalid date.
* public void setDate (int m, int d, int y) – set the instance variables to the values of the input parameters. For now, assume the values of the input parameters will not create an invalid date.

### **Software Testing – GVdateTest class (10 points)**

For this project, write a **GVdateTest** class that contains a main method that instantiates at least two objects of the GVdate class.

### **Sample Main Method**

We are providing examples of two different approaches to write a main method. Choose one of the approaches to do your testing.

**Main Method Example 1:** Using assert operation. See zyBooks chapter 13 – 13.2

Complete the main method given below if you choose this approach.

# public static void main (String [] args) {

# System.out.println ("Testing begings");

# //\*\*\*\*\*\*\*\*\*\* phase 1 testing \*\*\*\*\*\*\*\*\*\*\*\*

# // testing the default constructor

# GVdate today = new GVdate();

# assert today.getMonth() == 10 : "month should be 10";

# assert today.getDay() == 12 : "day should be 12";

# assert today.getYear() == 2020 : "year should be 2020";

# // testing constructor 2

# GVdate theDay = new GVdate(1, 10, 1995);

# // TO DO: complete the checks for month, day and year

# // testing setter methods

# //testing setMonth

# day.setMonth(8);

# // TO DO: complete the code to check for month

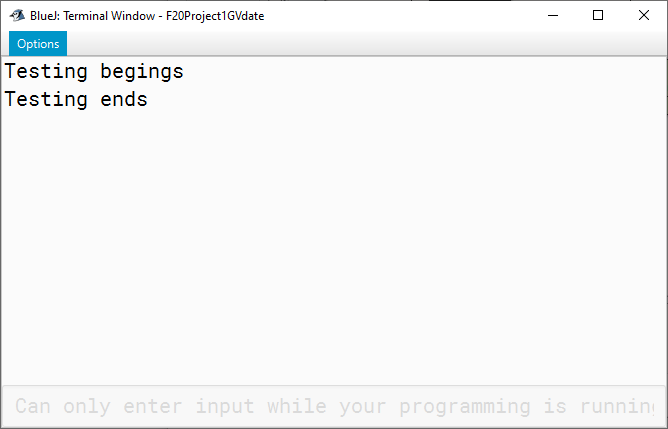
# // TO DO: finish testing setDay and setYear

# // TO DO: test the toString method

# System.out.println ("Testing ends");

# }

If your code passes all the test cases from the main method, your terminal window will show the following:



# If your code does not pass all the test cases, your program will stop and you will see the run time error message. For example:

# 

**Main Method Example 2:** Using ifs

Complete the main method given below if you choose this approach.

public static void main (String [] args) {

int errors = 0;

System.out.println ("Testing begings");

//\*\*\*\*\*\*\*\*\*\* phase 1 testing \*\*\*\*\*\*\*\*\*\*\*\*

// testing the default constructor

GVdate today = new GVdate();

if (today.getMonth() != 10){

System.out.println("month should be 10");

errors++;

}

if (today.getDay() != 12){

System.out.println("day should be 12");

errors++;

}

// TO DO: test the year

// testing constructor 2

GVdate theDay = new GVdate(1, 10, 1995);

// TO DO: complete the checks for month, day and year

// testing setter methods

//testing setMonth

theDay.setMonth(8);

// TO DO: complete the code to check for month

// TO DO: finish testing setDay and setYear

// TO DO: test the toString method

System.out.println("Errors: " + errors);

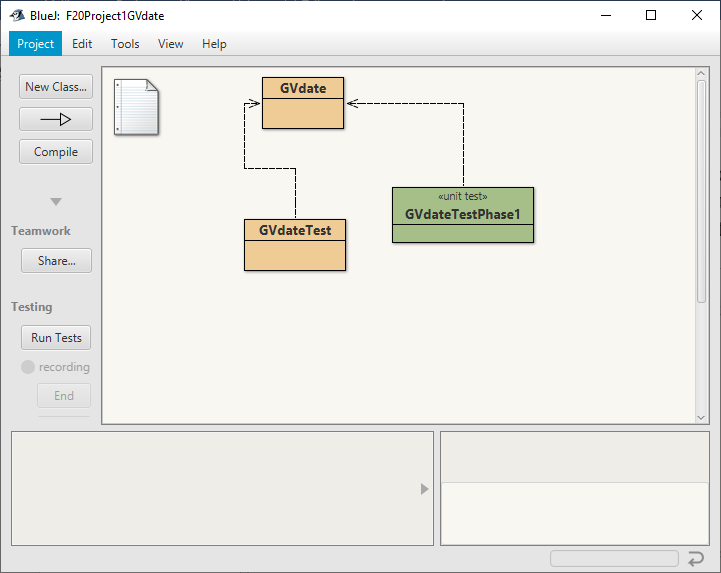
System.out.println ("Testing ends");

}

### **JUnit Testing**

JUnit is a unit testing framework for Java programming language. Three JUnit test files have been provided for you: GVdateTestPhase1.java, GVdateTestPhase2.java, GVdateTestPhase3.java. Follow these instructions to use the test files.

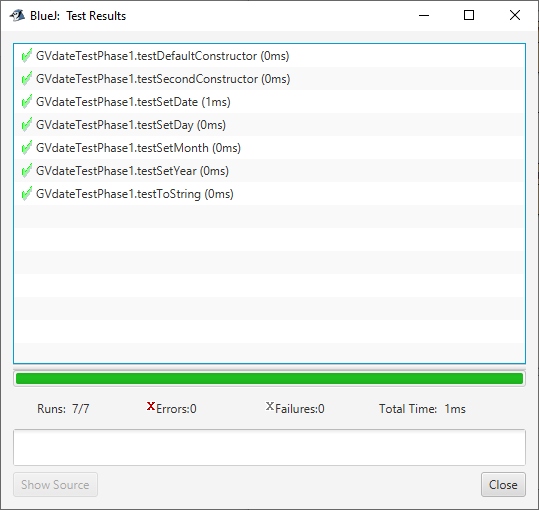
1. **Exact spelling is required for the name of the GVdate class and the headers of all the methods of the GVdate class.** The JUnit test classes will not compile if you change the name of the class or the method headers in the GVdate class.
2. Complete ALL requirements described above or some of the unit tests will fail.
3. Download GVdateTestPhase1.java to the folder that contains your GVDate.java file. Right click with the mouse and carefully choose where the file is saved. Be sure to maintain the correct filename, GVdateTestPhase1.java. Restart Bluej (if GVdateTestPhase1 does not show up on your BlueJ window). BlueJ should recognize GVDateTestPhase1 as a “<<unit test>>” file after you compile the file.



1. On the BlueJ window, click the “Run Tests” button.

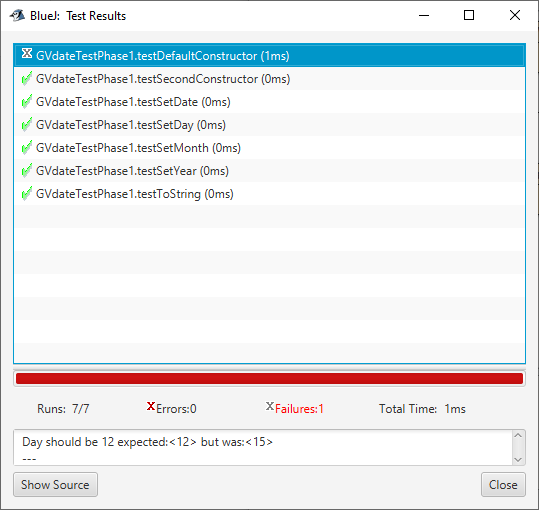
Example: JUnit test passes all the test cases

If your class passes all the test cases, the test results window will show a green horizontal bar and zero failures.



Example: JUnit test does not pass all the test cases:

If your class does NOT pass all the test cases, you will see a red horizontal bar and the number of failures. Click on a test result to show a longer description of the failure.



# Phase 2 (30 pts)

### **Write a third constructor.**

* public GVdate (String date)

This method parses the string passed as input parameter into individual pieces and converts them to integers. The date is contained in a string with values separated by slashes. The month and day can contain one or 2 digits, the year contains 4 digits. For now, assume the String entered as input parameter is correct.

Samples of the String date

"9/8/2020"

"09/8/2020"

"9/08/2020"

"09/08/2020"

Code example to parse the month:

int firstSlash = date.indexOf ("/");

month = Integer.parseInt (date.substring (0, firstSlash));

### **Write this additional method**

* public boolean isLeapYear(int y) – return true if the year passed as parameter is a leap year. Otherwise return false. Every year that is exactly divisible by four is a leap year, except for years that are exactly divisible by 100, but the centurial years (years divisible by 100) are leap years if they are exactly divisible by 400. For example, the years 1700, 1800, and 1900 are not leap years, but the years 1600 and 2000 are.

### **Private Helper Methods**

* private int getLastDayOfMonth(int m, int y)– return the last day of the month and year entered as input parameter. Remember to invoke the isLeapYear method when appropriate. If the month entered as input parameter is 2 (February) and the year passed as input parameter is a leap year this method should return 29.

* private boolean isDateValid (int m, int d, int y) – return true if day, month, and year entered as input parameters form a valid date, return false otherwise. To be able to write the logic for this method you need to know the number of days for each month of the year, whether a year is a leap year (leap years have 29 days for the month of February, not leap years have 28 days for the month of February). Remember to invoke the isLeapYear and getLastDayOfMonth methods when appropriate. Valid months are between 1 and 12 (inclusive) and consider a positive number as a valid year.

Examples:

* + Invalid dates:
    - 2/29/2019 – 2019 is not a leap year, the number of days in February is 28
    - 13/17/2019 – 13 is not a valid month
    - 11/31/2019 – the month of November only has 30 days
    - 6/23/-2019 – the year cannot be negative
  + Valid dates:
    - 2/29/2020 – 2020 is a leap year, February has 29 days
    - 11/30/2019 – nothing wrong, November has 30 days
    - 6/22/1992 – nothing wrong here.

### **Preventing User Error**

A good programming practice is to avoid, or at least to minimize, the effect of user errors.

* Modify each of the following methods to ignore invalid dates. Remember to invoke the isDateValid method when needed. The idea is to use the private methods to avoid repeating code.
  + Constructors – if a date is invalid, set the instance variables to the following values:
    - month: 10
    - day: 12
    - year: 2020
  + setDay - if the attempted new day, creates an invalid date, print an error message, and do nothing to the day.
  + setMonth - if the attempted new month, creates an invalid date, print an error message, and do nothing to the month.
  + setYear if the attempted new year, creates an invalid date, print an error message, and do nothing to the year.
  + setDate - if the attempted new date is invalid, print an error message and do nothing to the date.

# Update GVdateTest – Phase 2

Update your GVdateTest class to thoroughly test the new methods you just added. Test the third constructor, test the isLeapYear method, try to create invalid dates, and test the setters with invalid dates.

# JUnit Testing – phase 2

Run the GVdateTestPhase2.java that has been provided by your instructor and fix all the errors before continuing with phase 3.

# Phase 3 (30 pts)

* public String toString(int format)- The format parameter indicates the type of formatted date to be returned. This is an overloaded version of toString() that takes a parameter to indicate the format of the date to be returned.

format = 1 - return a formatted date string (for example "7/10/2019"). Could be one or two digits for the month and day and 4 digits for the year

format = 2 - return a formatted date string always displaying two digits for the month and day and 4 digits for the year (for example "07/10/2019"). You are not allowed to use any decision statements like if or switch. Hint: use the DecimalFormat class.

format = 3 - return a formatted date with the 3-character month description. So, the date 7/10/2014 would return the String "Sep 10, 2019". You are not allowed to use any decision statements like if or switch. You can do this by creating a single string with all the 3-character month abbreviations and using the substring method. The day could be one or two digits. The year has 4 digits.

Example: "JanFebMarAprMay…Dec"

format = 4 – return a formatted date with the month description. Use a switch statement to assign the correct month name (for example September 10, 2019). The day could be one or two digits. The year has 4 digits.

* public void nextDay( ) – advance the date one day. This may also require advances to the month and/or the year.

**Notes:**

Invoke the isLeapYear and getLastDayOfMonth methods when appropriate.

Instead of using literal numbers in your if statements, invoke the getLastDayOfMonth method

Examples:

4/10/2014 will advance to 4/11/2014

12/31/2014 will advance to 1/1/2015.

2/28/2013 will advance to 3/1/2013

2/28/2004 will advance to 2/29/2004 – note that 2004 is a leap year

* public boolean equals(GVdate otherDate) – return true if the month, day and year of the provided otherDate passed as a parameter matches the internal date - the instance variable date- Otherwise return false
* public void skipAhead(int numDays ) – advance the date by the number of days given as input parameter. Make sure the numDays is a positive number. Do nothing if the number of days entered as input parameter is negative. Instead of repeating code, use a loop to invoke the nextDay method as many times as necessary.

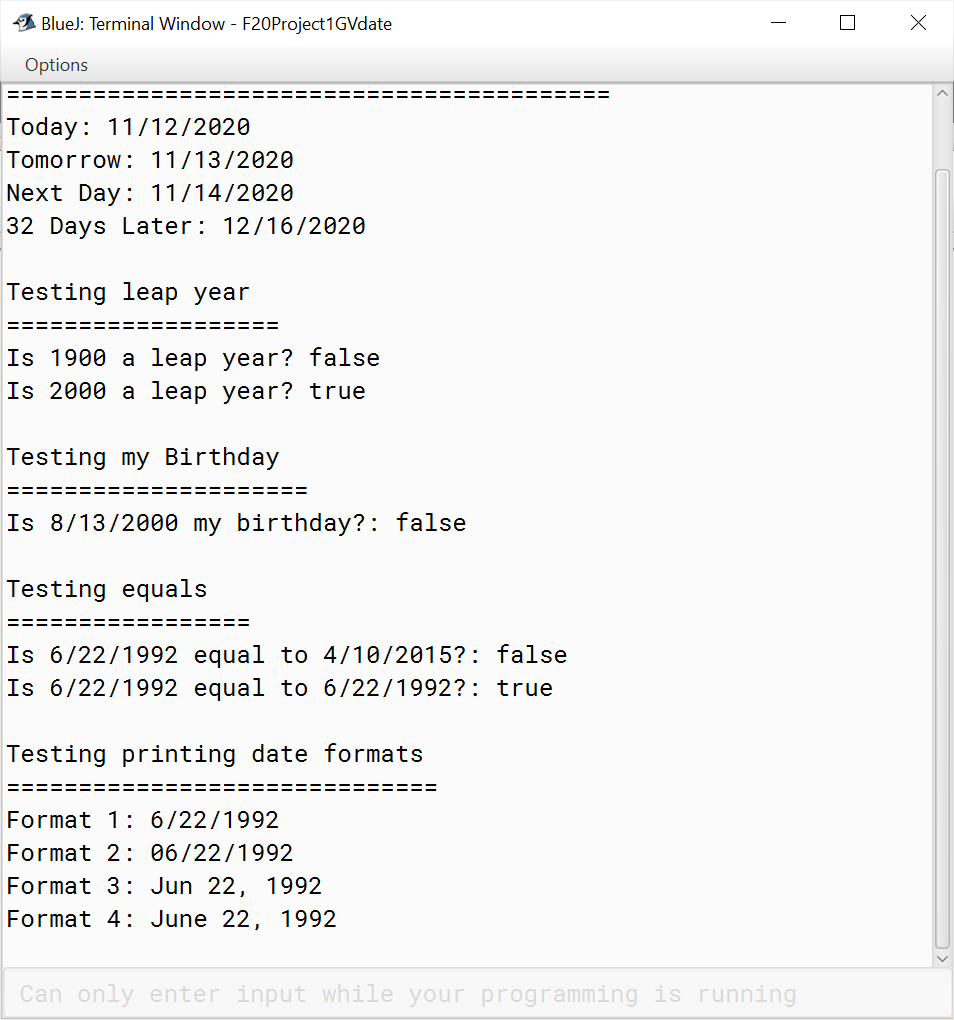
# Update GVdateTest – Phase 3

Update your GVdateTest class to thoroughly test the new methods you just added. Create two or more dates and test the following methods in your main method:

* nextDay
* skipAhead
* isLeapYear
* isMyBirthday
* equals
* toString method to format the date. Make sure you test all the different formats

### **Sample Output**

The following is provided as an example. Your messages can be more creative if they convey the correct information.



# JUnit Testing – phase 3

Run the GVdateTestPhase3.java that has been provided by your instructor and fix all the errors before submitting your work

# Grading Criteria

There is a 50% penalty on programming projects if your solution does not compile.

* Stapled cover page with your name and signed pledge. (-5 pts if missing)
* Project requirements as specified above. (90 pts)
* Elegant source code that follows the GVSU [Java Style Guide](http://www.cis.gvsu.edu/studentsupport/javaguide). (10 pts)

# Late Policy

Projects are due at the START of the lab period. However, you are encouraged to complete a project even if you must turn it in late.

* The first 24 hours (-20 pts)
* Each subsequent weekday is an additional -10 pts
* Weekends and university holidays are free days.

# Turn In

1. A document that includes:

* Cover page - Provide a cover page that includes your name, a title, and an appropriate picture or clip art for the project
* Signed Pledge (electronic) – The cover page must include the following signed pledge: "I pledge that this work is entirely mine, and mine alone (except for any code provided by my instructor). " In addition, provide names of any people you helped or received help from. Under no circumstances do you exchange code electronically. You are responsible for understanding and adhering to the [School of CIS Guidelines for Academic Honesty](http://www.cis.gvsu.edu/academic-honesty/).
* Time Card – The cover page must also include a brief statement of how much time you spent on the project. For example, “I spent 7 hours on this project from September 22-27 reading the book, designing a solution, writing code, fixing errors and putting together the printed document.”
* Sample Output – a printout of the BlueJ Terminal window after running the main method that shows a variety of the printed messages. You can copy and paste into the Word document that contains your cover page.
* A screenshot after you run the JUnit test class GVdateTestPhase3, showing you passed all the test cases.

1. Source code – DO NOT PRINT - upload to Blackboard the source code of the two classes.

* GVdate.java
* GVdateTest.java