**CIS 163 Project 1**

**A high*-end* ATM program**

**Due Date**

* At the beginning of lab on 31 January 2017, see the schedule, last page of the syllabus.

**Before Starting the Project**

* Review Chapter 1 – 6 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 *ATM Project***

* Create a package named: ***atmPackage*** (right click on ***ATMProject*** and select new/package)

**Step 2: Create a class named *ATM* that implements the following.**

For class properties, declare three instance variables:

* hundreds (integer)
* fifties (integer)
* twenties (integer).

There are no ten dollar bills and no one dollar bills.

For class behavior, implement the following methods (include any setters or getters that are needed). Unless otherwise stated, assume the input for step 2 will have no errors (i.e. no need to check for invalid data).

* public ATM() Default constructor that sets the ATM to zero.
* public ATM(int hundreds, int fifties, int twenties) A constructor that initializes the instance variables with the parameters.
* public ATM (ATM other) A constructor that initializes the instance variables with the *other* ATM parameter.
* public boolean equals( ATM other ) A method that returns true if ***this*** ATM object is exactly the same as the ***other*** ATM object, id est this.hundreds == other.hundreds, etetera. What is the difference between this method and the next method?
* public boolean equals( Object other ) A method that returns true if ***this*** ATM object is exactly the same as the ***other*** object (Note: you must cast the other object as a ATM object).
* public void putIn(int hundreds, int fifties, int twenties) A method that adds the parameters from the “this” ATM object. You may assume all of the parameter are positive (only for step 2).
* public void putIn (ATM other) A method that adds ATM ***other*** to the ***this*** ATM object.
* public void takeOut(int hundreds, int fifties, int twenties) A method that subtracts the parameters from the ***this*** ATM object. You may assume all of the parameters are positive, and there is sufficient quantities in the ATM (only for step 2).
* private void substract (ATM other) A method that subtracts ATM ***other*** from ***this*** ATM object.
* public ATM takeOut( int amount ) This method returns an ATM instance ATM( h, f, t ) where h \* 100 + f \* 50 + t \* 20 == amount. In other words,

if (h \* 100 + f \* 50 + t \* 20 == amount )

{

return new ATM( h, f, t )

}

(Maximize the largest currency in the returned ATM). The amount must be divisible by 10.

* + Example 1:

Given an ATM object ATM( 4, 8, 2 )

object.takeOut( 620 ) returns new ATM( 4, 4, 1 )

(Money left in the ATM object: ATM( 0, 4, 1 )

Hint: the correct combination for (h, f, t) must satisfy the conditions: h <= 4, f <= 8, t <= 2

* + Example 2:

Given an ATM object ATM( 0, 1, 2 )

object.takeOut( 170 ) returns new ATM( 0, 0, 0 )

(Money left in the ATM object: ATM( 0, 1, 2 )

Hint: no combination for (h, f, t), id est h <= 0, f <= 1, t <= 2 adds up to 170

* + Example 3:

Given an ATM object ATM( 9, 6, 2 )

object.takeOut( 60 ) returns new ATM( 0, 0, 0 )

(Money left in the ATM object: ATM( 9, 6, 2 )

Hint: no combination for (h, f, t), id est h <= 9, f <= 6, t <= 2 adds up to 60

* public String toString() Method that returns a string that represents an ATM instance with the following format: “1 hundred dollar bill, 10 fifty dollar bills, 1 twenty dollar bill”. Be sure to use proper pluralization. For example, bill or bills. See the output on page 4.

**Step 3: Software Testing: Using a JUnit test**

* Create a JUnit Test Case named ***ATMtest*** (right click on ***atmPackage*** and select new -- JUnit Test Case)
  + Cut and paste the file contents found on Blackboard in the ***Project 1*** folder under ***Assignments***.

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

**Step 4: Software Testing: Using a main program**

* The following is a start for some main programs. Your assignment is to add more Java main programs in order to test each ***ATM*** method separately and completely.

|  |  |
| --- | --- |
| **Example driver 1** | |
| **package** atmPackage;  **public** **class** AtmMain {  **public** **static** **void** main(String[] args) {    System.***out***.println("Testing ATM constructor\n" );    ATM s = **new** ATM(4,8,2);  System.***out***.println("new ATM(4,8,2) \n" + s.toString( ) );  System.***out***.println( s.getAmount( ) + " dollars" );  }  } | |
| **Sample output** | |
|  | Testing ATM constructor  new ATM(4,8,2)  4 hundred dollar bills  8 fifty dollar bills  2 twenty dollar bills  840 dollars |

|  |  |
| --- | --- |
| **Example driver 2** | |
| **ackage** atmPackage;  **public** **class** AtmMain2 {  **public** **static** **void** main(String[] args) {    System.***out***.println("Testing ATM putIn method\n" );    ATM s1 = **new** ATM();  System.***out***.println("ATM s1 = new ATM() \n" + s1.toString( ) );  s1.putIn(10,2,3);  System.***out***.println("s1.putIn(10,2,3) \n" + s1.toString() );  s1.putIn(0,0,0);  System.***out***.println("s1.putIn(0,0,0) \n" + s1.toString( ) );  }  } | |
| **Sample output** | |
|  | Testing ATM putIn method  ATM s1 = new ATM()  s1.putIn(10,2,3)  10 hundred dollar bills  2 fifty dollar bills  3 twenty dollar bills  s1.putIn(0,0,0)  10 hundred dollar bills  2 fifty dollar bills  3 twenty dollar bills |

|  |  |
| --- | --- |
| **Example driver 3** | |
| **package** atmPackage;  **public** **class** AtmMain3 {  **public** **static** **void** main(String[] args) {    System.***out***.println("Testing ATM takeOut method\n" );    ATM s2 = **new** ATM(2,1,3);  System.***out***.println("ATM s2 = new ATM(2,1,3) \n" + s2.toString( ) );  ATM temp = s2.takeOut(250);  System.***out***.println ("ATM temp = s2.takeOut(250)\n" + temp.toString() );  System.***out***.println("Remaining in ATM:\n" + s2.toString( ) );  }  } | |
| **Sample output** | |
|  | Testing ATM takeOut method  ATM s2 = new ATM(2,1,3)  2 hundred dollar bills  1 fifty dollar bill  3 twenty dollar bills  ATM temp = s2.takeOut(250)  2 hundred dollar bills  1 fifty dollar bill  Remaining in ATM:  3 twenty dollar bills |

|  |  |
| --- | --- |
| **Example driver 4** | |
| **package** atmPackage;  **public** **class** AtmMain4 {  **public** **static** **void** main(String[] args) {    System.***out***.println("Testing ATM save, load, and equals methods\n" );      ATM s1 = **new** ATM (5, 4, 3);  s1.save("pizza");    ATM s2 = **new** ATM();  s2.load("pizza");    System.***out***.println("s1.save(\"pizza\")\n" + s1.toString( ) );  System.***out***.println("s2.load(\"pizza\")\n" + s2.toString( ) );  **if** (s2.equals(**new** ATM(5,4,3)))  {  System.***out***.println ("\nThe save, load, and equals methods work!");  }  System.***out***.println(  "\nCreate many more test cases to prove " +  "the class is functioning correctly."  );  }  } | |
| **Sample output** | |
|  | Testing ATM save, load, and equals methods  s1.save("pizza")  5 hundred dollar bills  4 fifty dollar bills  3 twenty dollar bills  s2.load("pizza")  5 hundred dollar bills  4 fifty dollar bills  3 twenty dollar bills  The save, load, and equals methods work!  Create many more test cases to prove the class is functioning correctly. |

|  |  |
| --- | --- |
| **Example driver 5** | |
| **package** atmPackage;  **public** **class** AtmMain5 {  **public** **static** **void** main(String[] args) {    System.***out***.println("Testing ATM takeOut method\n" );    ATM s2 = **new** ATM(1,1,4);  System.***out***.println("ATM s2 = new ATM(1,1,4)\n" + s2.toString( ) );  ATM temp = s2.takeOut(110);  System.***out***.println ("ATM temp = s2.takeOut(110)\n" + temp.toString() );  System.***out***.println("Remaining in ATM:\n" + s2.toString( ) );  }  } | |
| **Sample output** | |
|  | Testing ATM takeOut method  ATM s2 = new ATM(1,1,4)  1 hundred dollar bill  1 fifty dollar bill  4 twenty dollar bills  ATM temp = s2.takeOut(110)  1 fifty dollar bill  3 twenty dollar bills  Remaining in ATM:  1 hundred dollar bill  1 twenty dollar bill |

|  |  |
| --- | --- |
| **Example driver 6** | |
| **package** atmPackage;  **public** **class** AtmMain6 {  **public** **static** **void** main(String[] args) {    System.***out***.println("Testing ATM takeOut method\n" );    ATM s2 = **new** ATM(6,9,2);  System.***out***.println("ATM s2 = new ATM(6,9,2) \n" + s2.toString( ) );  ATM temp = s2.takeOut(60);  System.***out***.println ("ATM temp = s2.takeOut(110)\n" + temp.toString() );  System.***out***.println("Remaining in ATM:\n" + s2.toString( ) );  }  } | |
| **Sample output** | |
|  | Testing ATM takeOut method  ATM s2 = new ATM(6,9,2)  6 hundred dollar bills  9 fifty dollar bills  2 twenty dollar bills  ATM temp = s2.takeOut(110)  Remaining in ATM:  6 hundred dollar bills  9 fifty dollar bills  2 twenty dollar bills |

Which approach for testing is better, Step 3 or Step 4?

**Step 5: Create the following additional methods in the ATM class:**

* public void save(String fileName)A method that saves the “this” ATM to a file; use the parameter filename for the name of the file.
* public void load(String fileName)A method that loads the “this” ATM object from a file; use the parameter filename for the name of the file.
* public static void suspend(Boolean on) A method that turns ‘off’ or ‘on’ any takeOut/putIn methods in. In other words, when true, prevents any takeOut/putIn method from changing (mutate) the state of the “this” object as it relates to the amount in the ATM.
* **An ATM instance from step 2; allow for an error in the input arguments for all constructors and methods, and throw an IllegalArgumentException exception if an error occurs.** For example,

“-120” is not a valid input string for a constructor in step 2 and an exception is thrown.

* + Sample code snippet many help:

if (amount < 0)

throw new IllegalArgumentException();

* + IMPORTANT: for the method takeOut, if you cannot make change or if suspend is true, then return null.

**The following will help you with reading from and writing to a file:**

The data file, as shown directly below, contains only one line of data. Listing 6.30 of your book shows how to use the Scanner class.

|  |
| --- |
| ANCHORAGE 256000 |

**The following code 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**(Exception error) {

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

}

}

**The following code 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 6: Software Testing: Complete the second part of the JUnit class named: ATMtest.**

* BE SURE to include JUnits test cases that show your ATM is properly throwing exceptions from step 5. **Note: one JUnit test per exception!**

**Step 7: Challenge Requirement, read chapter 6 in your book.**

* The following should only be attempted after all of the other requirements have been completed.
* Create a GUI front end to your project and create 3 ATMs with associated JButtons so that each ATM can be added to OR subtracted from. Also have a JButton that calls the suspend (Boolean on) method (only one JButton is need for all ATMs).
* Show the values for ***hundreds***, ***fifties***, and ***twenties*** on the display for each ATM.

Create a class called ***MyATMPanel*** that has a private inner class that implements ***ActionListener***. See chapter 6 in your book. Here is some help.

public class MyATMPanel extends JPanel {

private ATM ATM;

;

…..

public MyTimerPanel() {

ATM = new ATM(2,3,4);

….

}

private class TimerListener implements ActionListener {

public void actionPerformed(ActionEvent e) {

…..

--------------------------- 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 and the first 24 hours (-10 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: ATM Program Rubric.**

|  |  |
| --- | --- |
| **Student Name** |  |
| Due Date |  |
| 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>)   * 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   (one sentence after each @ param @return)   * Every method uses a /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* separator * Overall layout, readability, No text wrap * Using /\*\* … / for each Instance variable * Has many inner “inner” comments | | | 10 |  |
| **Steps 1 – 2: Basic Functionality** | | | 40 |  |
| **Step 3: JUnit test** | | | 8 |  |
| **Step 4: Main test** | | | 5 |  |
| **Step 5: Added functionality**   * **public void save( String fileName )** * **public void load( String fileName )** * **public static void suspend( Boolean on )** * **Allow for an error in the input for all constructors and methods** | | | 2  5  5  8 |  |
| **Step 6: More Software Testing** | | | 7 |  |
| **Step 7: Challenge Requirement** | | | 10 |  |
| **Total** | | | **100** |  |
| **How many hours did you spend developing this project?** | | | |  |
| **I received help from** |  | **for methods:** | |  |
|  |  |  | |  |
| **I gave help to** |  | **for methods:** | |  |
|  |  |  | |  |

**Additional Comments:**