GTBuzz

1 Introduction

Welcome to CS1331, Introduction to Object Oriented Programming! This assignment has two main parts to it. The first, called GTBuzz will have you write a simple program and experiment with errors. The second will be a simple GPA Calculator program to get you taking input from a user, writing conditional statements and performing calculations. Be sure to do both parts for full credit!

Make sure you read *the entire document*, there are some things even at the very end that you will not want to miss for this and future assignments!

Have fun :D

1.1 Java Installation

The programming language used in this course is Java. Make sure that Java is installed before running any programs. You can find instructions on how to install Java on Windows and Mac at the course site:

`http://www.cc.gatech.edu/~stasko/1331/started/index.html`

Note: Make sure that you install the Java JDK and NOT the JRE. The JDK is what we need to compile and build Java.

If you have any problems installing Java, then feel free to talk to any of the TAs via email, office hours, etc…

1.2 Text Editor/IDE Installation

After you install Java, you'll need to install a text editor or IDE to actually write your programs in. We don't have any real restrictions on text editors except that we want students to avoid using full-blown IDEs like Eclipse or IntelliJ until later on in the semester if you haven't used them before. If you don't know what Eclipse or IntelliJ are, no problem! We will introduce those environments later this year. Suggestions from the TAs for text editors/IDEs:

- Sublime Text (used by almost all the TAs, a real fan favorite, and it's really colorful)
- Notepad++ (a bare bones, basic, but solid editor for students just starting out)
- VIM/emacs (for those of you who have lots of experience, these are very powerful editors used widely in industry. Definitely not a first timers tool.)

- If you'd like to use an IDE, Professor Stasko recommends jGrasp to start. Much of his work in class will be done using jGrasp and you are more than welcome to do the same.

You should be able to find free (legal) installations for these around the Internet if you search around. Ask a TA for help if you need it!

2 GTBuzz

First, you will type Java source code using an editor. Next, you will compile the source code. Recall the purpose of the compiler and compilation step is to translate Java source code into bytecode and in the process check the code for compiler errors. Once free of compiler errors, the file is compiled and a file of bytecode is generated. The JVM then interprets and executes the bytecode. First things first - let's create a Java source file. In the editor of your choice, create a plain text file named GTBuzz.java. Type the following code into this file (don't just copy-paste as it may cause weird indentation issues):

```java
public class GTBuzz { //Begin class definition
      //Start of main method
      public static void main(String[] args) {
            System.out.println("I'm a ramblin' wreck from Georgia Tech!");
      } //End of main method
} //End of GTBuzz class definition
```

Please note that indention is also very important because it makes the code more readable and easy to debug. We will also be checking for good indentation style using our Checkstyle tool (see section 4 for more details).

2.1 Wooo, We Did It!

What exactly did we do...? Let's take a look!

- **public class GTBuzz {** This is the class definition. Every Java file starts with the class definition which HAS to match the same name as the file. For example, GTBuzz.java starts with public class GTBuzz {.

- **public static void main(String[] args) {** We'll get to the keywords later, but this is the main method header/ definition. The main method is the driving method. This means that the main method is the first method to be called and will start everything.

- **System.out.println("I'm a ramblin' wreck from Georgia Tech!");** Now we want to do something. System.out.println prints "I'm a ramblin' wreck from Georgia Tech!" to the screen. Note that the " symbols aren't printed when this happens.

- **} //End of main method** This is the matching curly brace to the main method header. The curly braces define a block of code, and in this instance the main method.

- **} //End of GTBuzz class definition** This is the matching curly brace to the class definition header for GTBuzz. Like the one above, this defines and ends the block of code that GTBuzz defines.

2.2 Compiling and executing a simple program

Now we want to compile this file into bytecode and then run it with the Java JVM.

- **Mac/Linux** Open up the Terminal app. To do this, browse out to the Applications folder then click on Utilities. You will find the Terminal app under this folder. Go ahead and open it.
- **Windows not 8** Go to Start and press run. Then type cmd into this box.
- **Windows 8** Press the Windows key, type "Command Prompt" and press enter

We now need to find your Java files with this handy terminal. Mac and Linux terminals usually start in the home directory for the logged in user. For example, a home directory of Bob is:

```
/Users/Bob
```

Windows command line would start in:

```
C:\Users\Bob
```

Now we need to get from the home directory to the Java files using the cd command (cd means "Change Directory"; directory meaning folder).

Example: Starting directory: /Users/Bob and our Java files are on the Desktop. So we need to do: cd /Desktop/

Command Line Protip: To see which files are in your current directory type "ls" on Linux/Mac or "dir" on Windows. This will help you figure out if you are in the right spot or not!

Now we are ready to compile and run our Java code. To compile Java files type javac GTBuzz.java, then if all goes well you will see another prompt. If text is outputted, then an error occurred. This will generate a GTBuzz.class file which is the bytecode we mentioned before. To run your application type java GTBuzz and press enter. You should see the following text. Don't type java GTBuzz.class.

Good job! You have finished your first Java program. The next part of the assignment involves changing things around and figuring out errors.

Note: If you are using an IDE like jGrasp, there will most likely be a button to compile and run your code. I won't give specific instructions here but you should be able to find these buttons in the documentation of the IDE.

2.3 Experimenting with Errors

We will be using the GTBuzz.java file that you just created and experiment with a few errors. For this, put your answers in a HW1Errors.txt file. For each error you need to do the following:

1. Start fresh from the GTBuzz.java file.
2. Make each change independently from the others.
3. Save the file as GTBuzz.java
4. Attempt to compile your code.
5. If you have an error message then note down two things in the HW1Errors.txt file.
   (a) Write, *in your own words* what you think the error is and how to fix it (one or two sentences only please!)
   (b) Take a look at what the message in the terminal window says. Copy and paste the line(s) of the error message (known as a "Stack Trace") that describes the error. Even if you don't know what the message means, you should be able to figure out which line explains what's going wrong. Ask a TA for help if you need it!
6. If no error was given, compilation was successful. Now try to run the program.
7. If there was no error then state that there is no error.
8. If there was an error running the program, note down the same two things from step 5 in the HW1Errors.txt file.
9. IMPORTANT: Fix what you changed in each experiment to be a fresh and working GT-Buzz.java file.
10. Repeat the process for all experiments.
11. When you have finished all of your experiments *fix all errors before turning it into T-Square* (see section 5 for more details)

2.4 Experiments

1. Change `public class GTBuzz {` to `public class GTbuzz {`
2. Insert the line: `int x = 3 / 0;` underneath `System.out.println`
3. Change `System.out.println` to `println`
4. Remove the first double quote in the string `"I'm a ramblin' ..."`
5. Change `"I'm a rambling wreck from Georgia Tech."` to `"How bout them dawgs."`
6. Remove the semicolon at the end of the `System.out.println` statement.

7. Remove the last curly brace at the end of the program.

Remember to revert to the original code, save, compile, and run to make sure the program is working before going on to the next experiment. **When recording possible errors, make sure to clearly indicate whether the error is a compile error or runtime error.**

3 GPA Calculator

3.1 Problem Description

We are going to follow the rules and guidelines of calculating GPA based on Georgia Tech's "Quality Point" system. Essentially, we multiply the number of credit hours by the value of the grade we receive for each class and divide the total value by the total number of credit hours we are taking. Who knows, maybe this calculator will be useful for you in the future :)

Written as a math formula, the total number of quality points earned is calculated by:

$4.0 * (numACreditHours) + 3.0 * (numBCreditHours) + 2.0 * (numCCreditHours) + 1.0 * (numDCreditHours) + 0.0 * (numFCreditHours)$

Where $numACreditHours$ is the number of credit hours earned with a grade of A and so on.

The sum of the credit hours is simply:

$(numACreditHours) + (numBCreditHours) +$
$(numCCreditHours) + (numDCreditHours) + (numFCreditHours)$

GPA is the total number of quality points divided by the sum of the credit hours. We won't worry about pass/fail courses :)

3.2 Solution Description

You'll need to prompt the user for the total number of credit hours received for each letter grade and return the result of the GPA calculation. **You can assume for this assignment, we will only make valid inputs to your program (positive integers)**. A sample output for your program is below (note the credit hour numbers are user input, GPA is returned by your program):

```
javac GPACalc.java
java GPACalc

A Credit Hours: 4
B Credit Hours: 4
C Credit Hours: 0
D Credit Hours: 0
F Credit Hours: 0
GPA: 3.5
```

4 Checkstyle

In this course, we use a program that automatically checks your code legibility and readability to make sure it adheres to Java programming standards. For every homework assignment, you will be expected to run Checkstyle and fix any errors that occur. **Try to get in the habit of fixing errors as you go along as opposed to all at the end**. As programs get bigger, the amount of Checkstyle errors get larger as well. The full style guide for this course that you must adhere to can be found by clicking **here** (note that the Checkstyle .jar and .xml files on that page is out of date, please only use the ones we provide to you on T-Square).

For the first homework assignment, we want you to learn how to actually run Checkstyle. *We will not be taking off any points for Checkstyle errors on this assignment*. However, if a Checkstyle error does occur, you are encouraged - but not required - to try and fix it (you might have errors about changing tabs to spaces or trailing whitespaces, see if you can find a setting in your text editor to do this for you automatically; Google is your friend!).

Come to us in office hours or post on Piazza if you have specific questions about what Checkstyle is looking for and how to fix Checkstyle errors.

First, make sure you download the `checkstyle-6.0-all.jar` and `cs1331-checkstyle.xml` from the T-Square assignment page. Then, make sure you put *both* of those files in the same directory (folder) as the `.java` files you want to run Checkstyle on. Finally, to run Checkstyle, type the first line into your terminal while in the directory of your Java files and press enter.

```
$ java -jar checkstyle-6.0-all.jar -c cs1331-checkstyle.xml *.java
Starting audit...
Audit done.
```

The message above means there were no Checkstyle errors. You can easily count Checkstyle errors by piping the output of Checkstyle through `wc -l` and subtracting 2 for the two non-error lines printed above (which is how we will deduct points). For example:

```
$ java -jar checkstyle-6.0-all.jar -c cs1331-checkstyle.xml *.java | wc -l
     2
```

Alternatively, if you are on Windows, you can use the following instead:

```
C:\> java -jar checkstyle-6.0-all.jar -c cs1331-checkstyle.xml *.java | findstr /v "Starting
    audit..." | findstr /v "Audit done" | find /c /v "hashcode()"
0
```

5 Turn-in Procedure

Submit your `GTBuzz.java`, `HW1Errors.txt` and `GPACalc.java` files on T-Square as an attachment. Don't forget to fix any errors in your `GTBuzz.java` file! Do not submit any compiled bytecode (`.class` files), the Checkstyle jar file, or the `cs1331-checkstyle.xml` file. When you're ready, double- check that you have submitted and not just saved a draft. Continued...

Keep in mind, unless you are told otherwise, non-compiling submissions on ALL homeworks will be an automatic 0. You may not be warned of this in the future, and so you should consider this as your one and only formal warning. This policy applies to (but is not limited to) the following:

1. Forgetting to submit a file
2. One file out of many not compiling, thus causing the entire project not to compile
3. One single missing semi-colon you accidentally removed when fixing Checkstyle stuff
4. Typos
5. etc...

We know it's a little heavy-handed, but we do this to keep everyone on an even playing field and keep our grading process going smoothly. Again, consider this your official warning :)

6 Verify the Success of Your Submission to T-Square

Practice safe submission! Verify that your HW files were truly submitted correctly, the upload was successful, and that the files compile and run. It is solely your responsibility to turn in your homework and practice this safe submission safeguard.

1. After uploading the files to T-Square you should receive an email from T-Square listing the names of the files that were uploaded and received. If you do not get the confirmation email almost immediately, something is wrong with your HW submission and/or your email. Even receiving the email does not guarantee that you turned in exactly what you intended.
2. After submitting the files to T-Square, return to the Assignment menu option and this homework. It should show the submitted files.
3. Download copies of your submitted files from the T-Square Assignment page placing them in a new folder.
4. Recompile and test those exact files.
5. This helps guard against a few things.
   (a) It helps insure that you turn in the correct files.
   (b) It helps you realize if you omit a file or files.[1] (If you do discover that you omitted a file, submit all of your files again, not just the missing one.)
   (c) Helps find last minute causes of files not compiling and/or running.

[1]Missing files will not be given any credit, and non-compiling homework solutions will receive zero points. Also recall that late homework will not be accepted regardless of excuse. Treat the due date with respect. The real due date is 8PM Thursday. Do not wait until the last minute!