Assignment 2

Points: 100

Programming Grading for this Assignment and the Course

This information is moved to the last two pages of this document to optionally conserve reprinting.

Part 1 of 4. (Points: 11 of 100)

Blackboard online Test based on Chapter 2, Exercise 2-4, and Page 77.

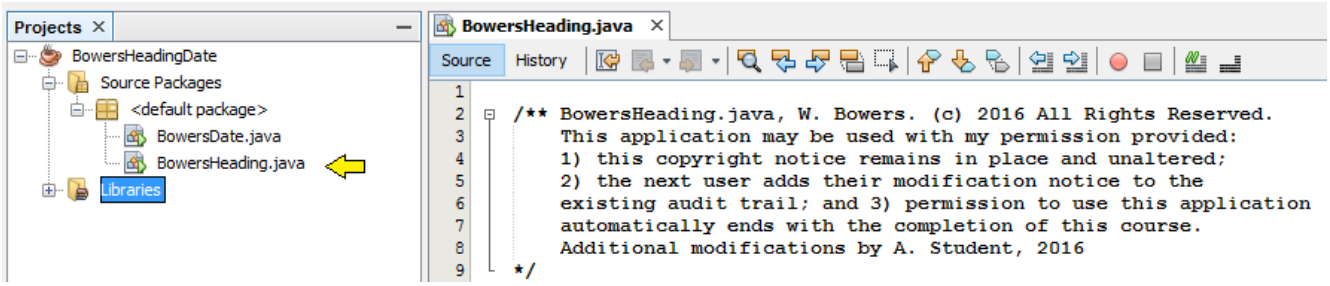
1 Follow the author's instructions completely, all six steps, because it will give you practice looking around the Java API and give you some check points to follow. There is nothing here that you need to turn in to me directly; but the practice will assist completing this Part 1.

2. In addition to following the author's directions, I am adding the requirement to look up the **DecimalFormat** class in the API, and perhaps keep it open on your desktop for ready-reference in order to answer the Blackboard online test questions you will find in the **Blackboard | Lessons | Week 2 | Week 2 Test Java API** link. You will have two (2) opportunities to take the test, and your highest score will automatically be recorded in the Blackboard Grade Book.

3. Some of the questions work with **associativity** and **math precedence**. You will find these topics covered in Blackboard in Learning Activity Links in Week 2, marked as such. These are programming topics you **must** know, and you will be tested about them.

Part 2 of 4. (Points: 11 of 100) Modify a Personal Heading Class

Do this Part 2 of the assignment **before** doing Part 4. Yes, Part 4. You need this Part 2 and Part 3 finished, working properly and ready to be used in Part 4 of this assignment. You are provided with **BowersHeading.java**, which resides within the BowersHeadingDate Project enclosed in the .zip file of the **Blackboard | Lessons | Course Home | Source Code Examples** Bowers Heading and Date | **BowersHeadingDate NetBeans Project** Your Starter Project. Download, extract the compressed file and open the project in NetBeans. Here's where you need to spend a lot (or a little) quality time with NetBeans to learn how to navigate the files.



A. Rename it to **YourLastNameHeading.java** (using your real last name, of course; for example, mine is appropriately named. Yours will be used with every assignment from this week forward and all through Java level 2. Mine, for example, is BowersHeading; yours, for example, might be SmytheHeading if your last name is Smythe. If not correctly named, spelled and capitalized,

you lose the points for this part of the assignment, as well as all future assignments until your file is correctly named.

How-to:
Click **once** on the **BowersHeading.java** class to select it. Assume your name is Samuel Smythe. Following steps:
Right-mouse
Submenu **Refactor**
Submenu **Rename** (top of the menu list)
In the Rename dialog box, replace Bowers with Smythe (your real last name instead), but keep the word Heading, no spaces, and the "H" remains capitalized. Also click\check the "Apply Rename on Comments" checkbox.
Click on the Refactor Button.
BowersHeading.java is now SmytheHeading.java

B. **Yours to-do**: Modify the opening programmer's comment to include (**add**) your name as the modifier of my original source code. You are required to, but have my permission to use the code, ***provided***: you **keep** my original citation and indicate you are the modifier of that code, to include your name, the current year at minimum. Do not delete my audit trail. For example, I expect to see (with your real name) added, changing Line 8 only: replace the "**A**" with your first name's initial followed by the period and a space; and replace "**Student**" with your real last name, a comma and the current year.

```
/**  BowersHeading.java, W. Bowers. (c) 2016 All Rights Reserved.
     This application may be used with my permission provided:
     1) this copyright notice remains in place and unaltered;
     2) the next user adds their modification notice to the
     existing audit trail; and 3) permission to use this application
     automatically ends with the completion of this course.
     Additional modifications by A. Student, 2016
*/
```

Anything more or less beyond the instructions provided is a gradable error.

C. **Yours to-do**: Other content in the class to change is using your names instead of mine: lines 16 and 17; your first name and last names instead of mine. The only changes to make are noted at the yellow arrow representing two lines of code: replace my first name with yours and my last name with yours; otherwise, I get the 11-points, and you do not.

D. **Yours to-do**: Immediately ahead of the main(…) method, add a **null constructor**: nothing in the argument (parenthesis), nothing in the (curly) brackets, all on one line. Look ahead to Pages 188 and 189 regarding constructors. Combine this reading with the additional reading and instructions provided in the **Learning Activity** link.

E. **Your information; learn only:** The main method that follows the null constructor has only

three lines of code.

1. Calls the getHeading(…) method, passing a String that you see to the getHeading(…) method.

2. Calls the comment( ) method to make a statement unique to this self-test. The reason for this simplicity is so you can test the heading program to make sure it displays all you want it to display, and it can also run as a stand-alone class.

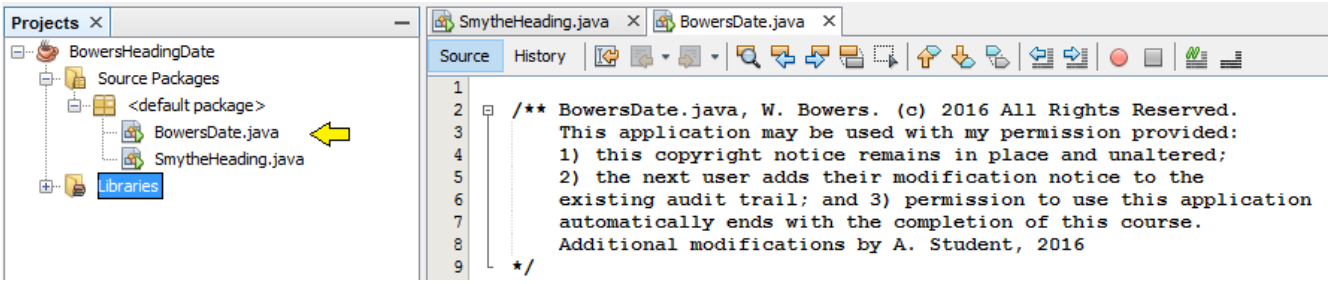
3. Calls the enterKey( ) method to complete the closing statements.

Exhibit A. Personal Heading class output. Any errors or omissions result in a total loss of Part 2's points.

```
run-single:


  William Bowers
  Personal Heading Class
  March 13, 2016

  Your personal Heading class uses println(...) method statements
  exclusively.  Except for testing this class's main(...) method,
  the getHeading() method will be used in every assignment to receive
  information from other applications.  All other methods will be used
  as needed throughout the quarter.

  The computer's getDate() and getTime() methods demonstrate alternative
  ways of coding quite differently from that used in your personal Date
  class.  You will need to know both ways of writing code.

  We will be using calls to your personal heading class in every
  assignment for the balance of the Quarter.

  March 13, 2016 10:56 PM
  Press the Enter key to continue.
```

Part 3 of 4. (Points: 11 of 100) Modify a Personal Date Class

Do this Part 3 of the assignment **before** doing Part 4. Yes, again, Part 4. You need this Part 3 finished, working properly and ready to be used in Part 4 of this assignment. You are provided with **BowersDate.java**, which resides within the BowersHeadingDate Project, which should already be open in NetBeans from just working on Part 2.



A. Rename it to **YourLastNameDate.java** (using your real last name, of course; for example, mine is appropriately named. Yours will be used with every assignment from this week forward and all through Java level 2. Mine, for example, is BowersDate; yours, for example, might be SmytheDate if your last name is Smythe. If not correctly named, spelled and capitalized, you lose the points for this part of the assignment, as well as all future assignments until your file is correctly named.

How-to:
Click **once** on the **BowersDate.java** class to select it. Assume your name is Samuel Smythe.
Following steps:
Right-mouse
Submenu **Refactor**
Submenu **Rename** (top of the menu list)
In the Rename dialog box, replace Bowers with Smythe (your real last name instead), but keep the word Date, no spaces, and the "D" remains capitalized. Also click\check the "Apply Rename on Comments" checkbox.
Click on the Refactor Button.
BowersDate.java is now SmytheDate.java

B. Modify the opening programmer's comment to include (**add**) your name as the modifier of my original source code. You are required to, but have my permission to use the code, ***provided***: you **keep** my original citation and indicate you are the modifier of that code, to include your name, the year at minimum. Do not delete my audit trail. For example, I expect to see (with your real name) added, changing Line 8: replace the "**A**" with your first name's initial followed by the period and a space; and replace "**Student**" with your real last name, a comma and the year.

```
/**   BowersDate.java, W. Bowers. (c) 2016 All Rights Reserved.
      This application may be used with my permission provided:
      1) this copyright notice remains in place and unaltered;
      2) the next user adds their modification notice to the
      existing audit trail; and 3) permission to use this application
      automatically ends with the completion of this course.
      Additional modifications by A. Student, 2016
*/
```

C. Change all other content in the class with your name instead of mine: line 19, replace my first and last name combination with your first and last name; no nicknames, initials or abbreviations here. You also need to make two changes on the long line 39, replace BowersDate with YourLastNameDate. The changes to make are noted at the yellow arrows representing three required replacements.

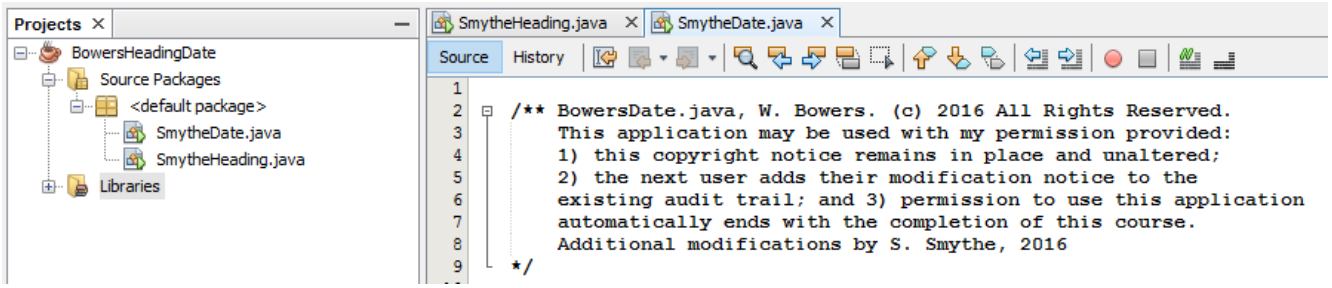
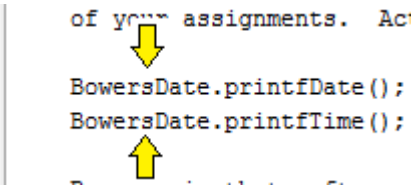


Exhibit B. Personal Date class output.

```
run:


  William Bowers
  Personal Date Class
  March 14, 2016

  The personal Date class uses printf(...) statements exclusively to
  include printfDate( ) and printfTime( ) method calls to display the
  computer's date and time on demand.

  From your applications, you will call these methods as follows, where
  my last name would be replaced by your last name here below and in all
  of y  assignments.  Actual usage and correct order of coding:

  BowersDate.printfDate();
  BowersDate.printfTime();

  Be c ain that, after your date and time displays, you guide the User
  of your application with the instruction to "Press the Enter key to
  continue." followed by a minimum of two (2) blank lines, as provided by
  a call to the enterKey( ) method.

  Monday, March 14, 2016  2:48:00pm
  Press the Enter key to continue.
```

Note: As you **examine the code** for both the personal heading and date classes, note the different styles of providing the date and time. The Heading class uses Date and DateFormat classes. The personal Date class uses **printf** formatting. You will need to have access to both styles for the balance of the course and into Java 2. Nothing you need to memorize; just know where to find it, what it looks like and how to use it as needed.

Final comment about this part of Part 3



Historically, statistically, every quarter, nearly 50% of each class (about 15 students) ignore this change and lose all 11-points. Of that number, every week thereafter, about 50% (again), or 7 students, continue to use an incorrect, unchanged citation. I will continue to assess points until it is repaired to my instructions. If the copyright continues to be violated by Week 4, it's a 100-point loss of points for the assignment.

Part 4 of 4. (Points: 67 of 100) Chapter 2, Exercise 2-3, Page 76

Modify the Invoice application; then we will modify it some more. Carefully follow all instructions, the author's and mine. Additionally, **Learning Activity 3** shows you how to add your newly-finished personal Heading and Date classes to Part 4's project.

Starter File: **ch02_ex3_Invoice**, as specified by the author in Step 1. You know, the author's files you were supposed to download in Week 1. I wasn't foolin'.

Download, compile, run and test the starter class. As shown in the text and with additional data entry shown here in Exhibit C, on the following page.

[Continued]

Exhibit C. **Author's starter** class (NetBeans) runtime, note I added a third input value (the text shows only two entries). Your program should be able to accept any number of entries, not just two or three, and any amount, not just that which is shown here. I always test the validity of your programming by using different counts and amounts. The reason for my paranoia is because past students have programmed specifically to examples shown in the text or in my exhibits; in other words, they did not program to the intent of the application. Please do not join their ranks.

```
Welcome to the Invoice Total Calculator

Enter subtotal:   100
Discount percent: 0.1
Discount amount:  10.0
Invoice total:    90.0

Continue? (y/n): y

Enter subtotal:   500
Discount percent: 0.2
Discount amount:  100.0
Invoice total:    400.0

Continue? (y/n): y

Enter subtotal:   1000.01      <-- Third Entry
Discount percent: 0.2
Discount amount:  200.002
Invoice total:    800.008

Continue? (y/n): n
```

[Continued]

Exhibit D. **Author's solution** class. Following the author's instructions, you will be coding to this level of output, followed by and **including** my additional, required modifications shown in **Exhibits** E1 and E2.

```
run:
Welcome to the Invoice Total Calculator

Enter subtotal:   100
Discount percent: 0.1
Discount amount:  10.0
Invoice total:    90.0

Continue? (y/n): y

Enter subtotal:   500
Discount percent: 0.25
Discount amount:  125.0
Invoice total:    375.0

Continue? (y/n): y

Enter subtotal:   1000.01
Discount percent: 0.25
Discount amount:  250.0025
Invoice total:    750.0074999999999

Continue? (y/n): n

Number of invoices: 3
Average invoice:    405.0025
Average discount:   128.33416666666668
```

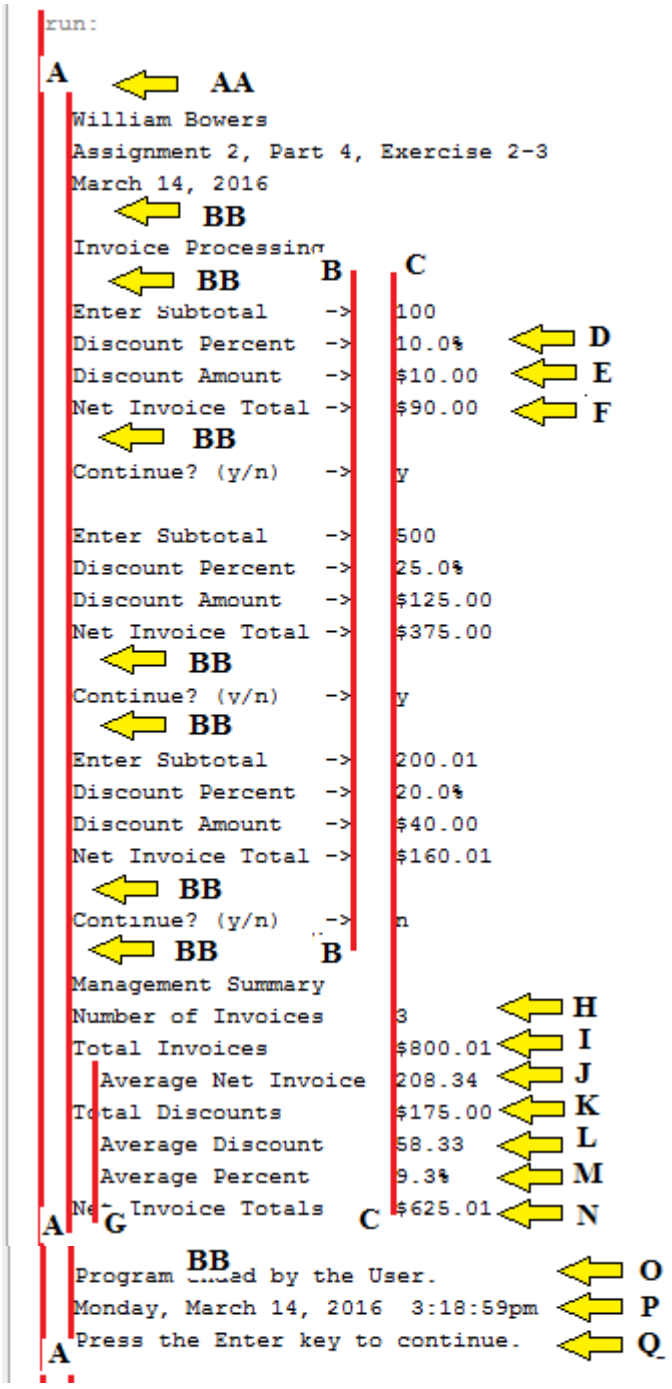
[Continued]

Exhibit E1. Additional, required modifications are listed below. The author's final solution is based on the set of instructions, Steps 1 through 4 on Page 76; combined with my additional requirements. When in doubt as to an instruction, code to this Exhibit in these situations. *An Exhibit is the equivalent of written instruction(s) and, therefore, gradable in every aspect.*

```
run:


  William Bowers
  Assignment 2, Part 4, Exercise 2-3
  March 14, 2016

  Invoice Processing

  Enter Subtotal     ->   100
  Discount Percent   ->   10.0%
  Discount Amount    ->   $10.00
  Net Invoice Total  ->   $90.00

  Continue? (y/n)    ->   y

  Enter Subtotal     ->   500
  Discount Percent   ->   25.0%
  Discount Amount    ->   $125.00
  Net Invoice Total  ->   $375.00

  Continue? (y/n)    ->   y

  Enter Subtotal     ->   200.01
  Discount Percent   ->   20.0%
  Discount Amount    ->   $40.00
  Net Invoice Total  ->   $160.01

  Continue? (y/n)    ->   n

  Management Summary
  Number of Invoices      3
  Total Invoices          $800.01
    Average Net Invoice   208.34
  Total Discounts         $175.00
    Average Discount      58.33
    Average Percent       9.3%
  Net Invoice Totals      $625.01

  Program ended by the User.
  Monday, March 14, 2016  3:18:59pm
  Press the Enter key to continue.
```

[Continued]

Exhibit E2. Formatting legend of Exhibit E1. When in doubt as to an instruction, code to this Exhibit in these situations. *An Exhibit is the equivalent of written instruction(s) and, therefore, gradable in every aspect.*



[Continued]

Legend of Formatting

AA Course required minimum and maximum of two (2) blank lines at the top of Form (ToF), first two lines of display.
BB Use of a single blank line; more or less than a single line is a gradable error.
A Course required minimum and maximum of two (2) blank spaces from the left of the display screen. Additional indentation, as shown in Legend G, below, is permissible when instructed.
B Right alignment of prompt pointers, followed by a minimum and maximum of **two** (**2**) **blank spaces** before the display of the user's input. Typically, 50% of students do not do this correctly.
C Left alignment of **all** user data entry and program output display. Again, typically, 50% error rate on the part of the student programmer. It's up to you to find out "how", practice at it.
D Minimum **and** maximum of **one** (1) decimal place followed by a percent symbol.
E Minimum **and** maximum of **two** (2) decimal places with the use of a dollar sign.
F Minimum **and** maximum of **two** (2) decimal places with the use of a dollar sign.
G **Additional two** (2) space indentation for **all** of the Management Summary **Averages.**
H Number of Invoices, note the spelling and capitalization; no decimal point or decimal places.
I Total Invoices, note the spelling and capitalization; minimum **and** maximum of two decimal places; dollar sign **is** used.
J Average Net Invoice, note spelling and capitalization; minimum **and** maximum of two decimal places; **dollar sign** is **not used**.
K Total Discounts, note the spelling and capitalization; minimum **and** maximum of two decimal places; **dollar sign is used**.
L Average Discount, note the spelling and capitalization; minimum **and** maximum of two decimal places; **dollar sign** is **not used**
M Average Percent, note the spelling and capitalization; minimum **and** maximum of **one** decimal places; **percent sign is used** following the decimal place.
N Net Invoice Totals, note the spelling and capitalization; minimum **and** maximum of two decimal places; **dollar sign is used**
O Termination message, note the spelling, capitalization and punctuation as shown: "Program ended by the User." Without the quotes, capitalized P in Program and U in User, and the sentence ended with a period. 80% typical error rate here, please be careful.
P Date and time are both displayed on one separate line by calling on the personal date's printfDate( ) and printfTime( ) methods. You have examples of how to do this in the personal Date class code.
Q The "Press the Enter key to continue." message is automatic; that is, already programmed when calling the printfTime( ) method.

Required Modifications Specifications

1. Add your personal Heading and personal Date classes to this and all projects. There are two ways you can accomplish this.

    a. Using your new personal heading and date classes created in Parts 1 and 2, copy just the .java files, **YourLastNameHeading.java** and **YourLastNameDate.java** classes into the

src folder of your NetBeans project. NetBeans will take care of the rest and create the .class files when it compiles and builds.

b. Or, follow the tutorial\procedure for adding an existing file into another existing project: **Blackboard | Lessons | Course Home | NetBeans Tutorial, Add an Existing File to an Existing Project**.

Yours will be using your real last name	**Mine, for example is as follows**
YourLastNameHeading.java	BowersHeading.java
YourLastNameDate.java	BowersDate.java

You needed to complete Parts 2 and 3 so you can use them here in Part 4; if you did not, then start over and follow directions. There are reasons for directions; kind of like course prerequisites.

2. For the sake of less code writing and simplification, we will assume the user performs accurate data entry. We will work with exception handling in a later week.

3. Be sure to **add** your name and year modification citation to the opening programmer's comments. Again, another 20% of you may overlook this 8-point-loss opportunity. The citation should appear as follows with your real initial and last name in place of the emboldened text. Remember that citations are always the first lines of code in the JavaDoc format of a programmer's comment.

```
/**  Murach, J. ( 2011).  Murachs Java Programming, Training and
     Reference, 4th Edition,  Fresno, CA: Mike Murach & Associates, Inc.
     Modifications by W. Bowers, 2016
     Additional modifications by A. Student, 2016
*/
```

Replace "**A**" with your real first initial; replace "**Student**" with your real last name. Anything other than shown above and deviated from the instructions is unacceptable and a gradable error.

4. Application output\display starts a minimum and a maximum of two blank lines down from the top of the display screen. This is also a course requirement. Your new personal heading class should take care of that for you. If it doesn't, fix the personal heading class, not this application.

5. **All** margins are two (2) spaces off of the left margin of the output screen. This makes for cleaner reading of the output, and it's a course requirement. Additional indentation as shown on the Average's lines in the Management Summary is required as shown.

Also note, all colons are replaced in this application with pointers. Pointers, or arrows for the less programming-oriented, are a combination of a hyphen (not underscore) and a greater-than symbol (arrow). Additionally, note they are all perfectly aligned one above the other for the entire output in the data entry and management summary areas. Again, do not continue to use

colons, at 8-points each. I'm nagging because typically, 50% of you will miss this change and the alignment, as obvious as it is; another opportunity to improve a statistic. Huzzah!

6. Call your new personal heading class's getHeading(…) method from InvoiceApp's main(…) method, and pass the String "Assignment 2, Part 4, Exercise 2-3" in the argument (parenthesis). For example, mine would be coded, **`BowersHeading.getHeading("Assignment 2, Part 4, Exercise 2-3");`** You already have an example in the previous sentence and in your personal heading class.

7. Change the "Welcome to the Invoice Total Calculator" banner line to a title line, with just "Invoice Processing", without the quotes, of course. Statistically, historically, 80% of you will ignore this and its proper spelling, lack of punctuation and use of capitalization; let's try to improve on that "stat".

8. Before displaying your output, here's a **Q & A** session.

   Question. Do I have to display the output as shown in Exhibits E1 and E2, above; to include alignment, margins, use of blank lines, additional indentations, dollar signs, percents as shown, and everything as shown in the redisplays and management summary; to also include spelling, appropriate capitalization, use of arrows (pointers to you, now), lack of colons; and anything classified as "etc."?

   Answer. ***Yes***. You must code to specifications explained and the finished Exhibits shown. It is your visual blueprint. But because I cannot anticipate "infinity-and-beyond" interpretations of the instructions, when an instruction is unclear, code to the final Exhibits shown above; you would be graded against this anyway. These are screen prints of my final, live code in action.

   a. Use the **DecimalFormat** class (examples are found in the **NumberFormatting** class provided in Week 2's Blackboard link.) ***for anything dealing with currency***. You will need two instances of the DecimalFormat class.

      1) When needing a dollar sign, use the pattern **`"$#,##0.00"`**; this would accommodate the output for the Total Invoices, Total Discounts and Net Invoice Totals lines.

      2) When not needing a dollar sign, use the pattern **`"#,##0.00".`**

      ```
      Hints:
      DecimalFormat cf = new DecimalFormat("$#,##0.00");
      DecimalFormat nf = new DecimalFormat("#,##0.00");
      ```

   b. Use the **NumberFormat** class **only** to display the ***percent output*** (Chapter 3, Pages 94 and 95). Use the getPercentInstance( ) method, followed by the setM**ax**imumFractionDigits(1) ***and*** setM**in**imumFractionDigits(1) in order to redisplay the Interest Rate entered by the user with a minimum and a maximum of **one** (**1**) decimal place although the user may enter up to any number of decimal places.

```
Hints:  NumberFormat pf = NumberFormat.getPercentInstance();
        pf.setMaximumFractionDigits(1);
        pf.setMinimumFractionDigits(1);
```

9. Course requirement: all user prompts must leave the cursor on the line of the prompt for user entry. To move the prompt to a lower line is unprofessional and a programming error and worthy of an 8-point loss for **each** erroneous prompt. Also, and a course requirement, a **minimum** of two (2) blank spaces are needed between the end of a prompt and the beginning of the user's data entry.

   For example, where the 500 shown is the input from the user:

   Acceptable `Enter Subtotal ->  500`
   Not Acceptable `Enter subtotal-> 500`

   Two things wrong with the Not Acceptable version: first, Subtotal should have a capital "S"; second and there is only one space between the arrow and the value of 500. Yes, I expect that much attention to detail.

10. When data entry is completed, provide a **Management Summary**, as shown, to include all of the accumulated and recalculated data shown in Exhibit E. Be very careful of **capitalization** and **spelling**, distance between the description and the value it's describing, the perfect left alignment with the data entry sections, and the order in which they are presented. For example, Number of **invoices** is incorrect; Number of **Invoices** is correct. Use variables to display calculations, do not hard code the values; I check code carefully, and I use a different number of data entries as well as different data entry values to test your programming.

Management category	**Variable or Variable Formulae**
Number of Invoices	invoiceCount
Total Invoices	invoiceTotals (new global variable created to handle grand totals)
Average Net Invoice	invoiceTotal / invoiceCount
Total Discounts	discountTotal
Average Discount	discountTotal / invoiceCount
Average Percent	(discountTotal / invoiceCount) / invoiceTotal
Net Invoice Totals	invoiceTotal

   If calculations of calculations are off by a rounded penny (or two), for the scope of this course, I do not care. More than twopence (tuppence to you Mary Poppins fans), you may have a problem.

   You may have to move the declaration of the **total** variable ahead of main as global.

   Q. What's a global variable?

   A. Any variable declared and\or initialized ahead and outside of the first method (any method or the main(…) method) in a class. It makes the variable visible and usable in all subsequent methods in the class. This is applicable to all programming languages, not just Java. If a variable is declared inside of a method, any method, then it is considered a

local variable.

11. As shown, display a "Program ended by the User." statement (watch spelling, punctuation and case sensitivity, it is a sentence, not a title line, and User is a proper noun) on its own line, then call both your personal **date** class's **printfDate**( ) and **printfTime**( ) methods to display the date and time information on the **next line**, as shown. You already have an example of this in your personal date class because it is self-testing.

12. The "Press the Enter key to continue." message, followed by two (2) blank lines is already built into your personal Date class's printfTime( ) method; nothing additional for you to do here except code it. But you say it doesn't clear the runtime view in NetBeans. That is correct, but two things: first, it's an instruction (a "Because-I-said-so" nonsense); second, only NetBeans keeps the runtime screen in view until you 'x' it away. All other IDE's close the runtime screen and return to the source code page. Microsoft products won't budge until you do press the Enter key. You are making your program compatible, friendly, transportable and other such adverbs, gerunds, dangling participles and adjectives, as well.

Submission Requirements

Compress your **entire**, primary NetBeans **InvoiceApp** folder to a **.zip** file and attach it to a **student email** addressed as follows. Please do not send me pieces of the project, I must have all of it, or you will not receive a grade for the assignment. What I cannot work with, I cannot grade.

To: **wbowers@mccneb.edu**
Subject: **YourLastName | Info 1521 WW | Assignment 2, Part 4**

PS, a Heads-up

1. This is the last time (week) we will use .zip files. Beginning with next week's Week 3 assignment, we will submit our NetBeans folder in a **.jar** file. Jar stands for **J**ava **Ar**chive File. Start studying and practicing the ***Jar Files Tutorial for NetBeans*** in **Blackboard | Lessons | Course Home | Tutorials for Java**. I wanted to give you a small amount of time to adapt to Java before adding to your knowledge. After Week 2, ***no one*** will be exempt from using Jar files for the balance of this quarter and into Java 2.

2. In order to have perfect as well as flexible output formatting, we will also begin using printf(…) statements in Week 3. More easing into the language. **Blackboard | Lessons | Course Home | Tutorials for Java | Printf Tutorial**. Beginning with Week 3, printf(…) statements will be used periodically throughout the quarter and into Java 2

Reminder

When in doubt as to an instruction, whether I made an error, or it may be ambiguous, or you just don't like the way I tell a tale or try to assist, code to the finished exhibits. This saves time and email exchanges in many small situations. In this assignment's case, you would be coding to **Exhibits E**; for your personal Heading class, **Exhibit A**, and **Exhibit B** for personal Date class. You will never be penalized for an error on my part, only errors on your part.

Java Computer Programs: Grading for this Assignment and the Course

The following is excerpted from the Syllabus and, therefore, applicable to this assignment as well as the whole course. You will see this message each week as a reminder; it will be added-to in the near future, so don't lightly skip over it.

1. A **100-point loss**. Got a question? Ask me directly. Under no circumstance may you use the respective courses' student email addresses:

   For In-class: info_1521_sn@mail.mccneb.edu, or
   info_1531_sn@mail.mccneb/edu; or
   For Online: info_1521_ww@mail.mccneb.edu, or
   info_1531_ww@mail.mccneb.edu

   .

   In order to mass communicate or inquire with\of\to your colleagues. The appropriate address is reserved for the instructor to communicate with the entire class at once with College or course related information. No exceptions.

2. Once submitted, a program may not be updated, added-to or revised. You only have one opportunity to submit your work for grading. Additionally, **I do not review or approve work-in-progress**; I only grade your final solution. Seeking pre-approval is not an acceptable programming skill.

3. Working programs will be assessed **eight** (**8**) **point loss** *for each incident* of an error, omission or misspelling of an instruction and/or output display. This is standard for all of my courses; **strong** incentive to do a thorough job.

4. An **8-point loss** will be assessed if your opening programmer's comment with appropriate citations is missing, inaccurate, out of date or incomplete, whether you modify the file or not. Not adding the opening programmer's comment to include your name in this and **every** program you write or modify is an automatic **loss of 8-points per file** of the assignment. You must provide appropriate program audit trail information and cite your sources for the code; otherwise, it's a potential for plagiarism. For more citation information and recommended *minimum* verbiage, see the applicable Blackboard links in **Lessons** | **Course Home** | **Mandatory Java Citations**

5. Non-working programs are worth zero (0%) points. I do not buffer or support a failed program with points for "effort". I know what effort it takes, all of my programs work. Nominal changes such as returning starter files with just a name added or a couple of lines of modified code, returning exactly what you may have been provided, or willful destruction of an assignment is an absolute 0%. The instructor makes this determination; and it is final, I do not re-grade.

   At the beginning of each new week, you can examine the previous assignment's solution code at your convenience in the "**Last Assignment Solution**" link in **Blackboard**. Take the opportunity

to learn how the code works compared to your work, and how the solution compares to the assignment's instructions. These are excellent sources of live, working Java code, and it is recommended that you not only study it to improve your technique, but to also build a library of how-to's for future reference.

6. **50-point assessment**. Personal programming files.
   a. YourLastNameHeading.java class
   b. YourLastNameDate.java class

   The above are support applications that are also self-executing for testing purposes; but they are primarily used as support files for **every assignment in the course**, beginning with Assignment 2, and\or as directed by the instructor. They are copyrighted with all rights reserved by the instructor, and their use is limited to the instructor's instructions and for this course only. To not use them as directed is a severe copyright infringement.

   The intent of these additional classes is to practice the use of object oriented programming and the ability to coordinate multiple files in a project. When we begin using them, and you will be advised when, your personal Heading and Class files must not have any assignment code in them, nor can your assignment replace the use of your personal versions of these files.

7. Late assignments, even by a minute, lose all **100-points**, without exception. The assignment will not be reviewed, just graded with a zero and filed; all excuses and reasons will be ignored. Refer to the Syllabus for the definition of "late".