

## CISC 124: Introduction to Computing Science II Winter 2017

### Assignment 5: GUI with Java Swing

**This assignment is due by April 9, 2017 @ 11:59 PM.** You may submit up to a maximum of 24 hours late (April 10, 2017 @ 11:59 PM) for a 10% penalty.

General questions about this assignment (administrative details, requirements, etc.) should be directed to the onQ Discussion "Assignment 5". Please do **NOT** post any Java code or pseudo-code on onQ. If you need help with programming, please see a TA during lab hours or use the instructor office hours (posted on onQ).

Attached you will find the description of **TWO projects**. The first project is an extension of Assignment 3. The second project is a small game. **Choose one of the projects to implement. You do not have to implement both projects.**

This assignment will allow you to be creative and design your own interfaces. The project description will tell you how the program needs to work and the minimum required GUI components which have to be integrated into the interface.

#### Working in Groups

If you decide to implement the 2nd Project (the game), you may work alone or in a group of 2 students. If you work in a group, you do so at your own risk. If the other member of your group gets sick or backs out for any reason, you are still responsible for handing in an assignment. Please see additional instructions below for submission of group assignments.

#### Some general remarks:

- Both projects can be implemented using just the GUI components and concepts discussed in class. If you would like to use other components, which are available in the Java Swing library, you are allowed to do so. However, please keep in mind that the TA's might not be able to provide you with help in this case.
- I'm aware that most IDE's have a GUI builder integrated, which allows you to simply place components in a frame by drag and dropping them. Using the GUI builder will not allow you to practise the use of the Swing and AWT components, particularly the different LayoutManagers. It will not prepare you for the exam. Don't use them!
- If you use icons and you would like us to see these icons, don't forget to attach the image file for the icon to your assignment. If you download images for your icons from the internet, don't forget to check whether there is a copyright infringement. There are many places in the internet which offer free use of icon images; please use these ones.
- We are going to grade this assignment primarily by using it. Therefore, it is **VERY** important that you submit a program which compiles without errors. If you have problems getting your program to compile, please use the lab hours. The TA's will be

happy to help you. On this note, your GUI should be sufficiently self-explanatory so that we can figure out how to use it without a manual. Both projects are simple enough that this should not really be a problem. If you feel that an explanation is necessary, you may use a JLabel with instructions, or add tool tip text.

- We will not check programming style in this assignment. By now I hope that “your fingers” will automatically write code which follows good programming style requirements. Please keep doing it. It will be required in other courses throughout your studies AND in any programming-related job.

#### **Submission of your assignment:**

- If you need to submit more than one .java file, combine them into a .zip file.
- Your .zip file should contain nothing but .java files — and perhaps image files such as .jpg or .png if you want to use icons (not required).
- Your submission should be a complete program. That means it should also contain the class containing the static main method.
- Your .zip file should not contain any folders, which means your program must be one package only.
- **For group submissions:** Only one of the group members should submit the assignment. The onQ dropbox gives you the opportunity to add a comment to each submission. Add the name of your teammate in this comment section. This will make grading easier.

#### Tips for how to approach this assignment:

- Start by making a design for your GUI on paper.
- Design your program. Identify which layout managers you would need to use to generate the designed GUI. Are there components which can be encapsulated into its own class? Which methods should this class have? Both projects are easy enough that your class structure will be very simple - no complicated inheritance or abstract classes- but it offers you a perfect opportunity to practise your understanding of object-oriented programming. However, we are not going to mark the assignment with regard to the project design. You do not need to document this step.
- For the implementation, you can follow two strategies:
  - 1) Make it work first and make it pretty later: Start with implementing a rough GUI which has all the functionality without spending the time to make it look good. After you make sure that everything works correctly, improve the appearance of your program.
  - 2) Make it look good first and make it work later: Start with implementing your GUI the way you designed it. After this, add the functionality to the GUI components.
- **Attention Mac Users!!!** The combination of Java 8 and the newer MacOS version has an error in setting the background colors for most/some GUI components. You may find that some components don't use the new background color or that the background color changes while the application is running. You can solve this problem by switching the “Look-and-feel” from the default value “System look and feel” to the Java “cross

platform look and feel". Just add the following lines of code at the beginning of your main method:

```
try {
    UIManager.setLookAndFeel(
        UIManager.getCrossPlatformLookAndFeelClassName());
} catch (Exception ex) {
    System.out.println("Not found.");
}
```

### Submitting as a group

- In your main class file (the one which contains the main method), both of your names and/or student IDs need to be in the header comment line of the file. If your program has more than one java file and you split the implementation of the classes between team members, only put the name and/or student ID of the person who implemented the java class. If you both worked on the implementation of this file, put both of your names in the header line. We will not count lines to see if both of you have done the same amount of work. This is up to your team management.
- Only one member of the group should submit the assignment. The onQ dropbox gives you the opportunity to add a comment to each submission. Add the name of your teammate in this comment section. This will make grading easier.
- Even if you split the work for this assignment and use the header comment lines in each file to identify who did which work, **you will both get the same mark for the assignment.** It is important that you both test and debug the final program together before submitting it.

### Grading Scheme:

- Functionality: 10 points
- GUI: 15 points
- Summary: 25 points

### Academic Integrity:

Any of the following actions constitute a violation of Queen's Academic Integrity standards:

- Copying another student's work, even if you modify it afterwards
- Allowing another student to copy your work
- Allowing anyone else to write code for you for this assignment (except for team members)

Please consult Queen's Academic Integrity webpage for more information.

**HAPPY PROGRAMMING!!**

## 1. Project: Market Checkout GUI

The goal in this project is to implement a graphical user interface for the market checkout system from Assignment 3. Attached to this assignment you will find a zip folder *MarketClasses.zip*, which contains slightly modified versions of the Java classes you developed in Assignment 3: *MarketItem.java*, *Produce.java*, *PreparedMeal.java* and *Checkout.java*. There is also a helper class *Money.java*, which contains some static methods to round and format money values. There are some small differences in the attached classes compared to the classes developed for Assignment 3. For example, the *PreparedMeal* class has a new attribute *amount* from type *int*. The customer can now purchase more than one prepared food item. The *Checkout* class still has an attribute *items* from type *MarketItem[]*. However, unlike the implementation in Assignment 3, the *Checkout* class can now add single items to this list using the *addItem* method. A possible use of this method could be:

```
Checkout myCheckout = new Checkout();
Produce p = new Produce("Tomatoes", "A Farm", 2.5, 1.5);
PreparedMeal pm = new PreparedMeal("Falafel", "A Kitchen", 2.5, 2);
myCheckout.addItem(p);
myCheckout.addItem(pm);
```

The *Checkout* class now also has the following three instance methods:

```
public String getTotalPriceString();
public String getTotalTaxString();
public String getTotalAmountString();
```

Each of these methods returns a *String* of the corresponding value in a "money format".

In this assignment, ***you are allowed to modify any of these classes if you need to***. However, if you modify one or more of the attached classes, do not forget to attach them to your submission.

The task is to implement a program which allows the user to type in information about a market item (name, vendor, price and amount) and to add this item as produce or a prepared meal item to the checkout list. The program should display a list containing information of all items which were entered. The program should also display the total price for all items, the total tax, and the total amount (price + tax). This information should be updated each time the user adds a new item.

The GUI should have at minimum the following components:

- A text field for the item name
- A text field for the vendor name
- A text field for the item price
- A text field for the amount
- A button to add a produce item
- A button to add a prepared meal item
- A text area in which a list of all added items are displayed
- A label (or text field) displaying the total price for all items in the list
- A label (or text field) displaying the total tax for all items in the list
- A label (or text field) displaying the total amount (price + tax) for all items in the list

Your program should have the following functionalities implemented:

- The user should be able to type in information about name, vendor, price and amount of an item.
- When the user clicks a “Produce” button, an object from type *Produce* should be created using the information the user filled into the above-mentioned text fields. The new object should then be added to the Checkout’s *MarketItem* list.
- When the user clicks a “Prepared Meal” button, an object from type *PreparedMeal* should be created using the information the user filled into the above-mentioned text fields. The new object should then be added to the Checkout’s *MarketItem* list.
- When a new *Produce* or *PreparedMeal* object was created, the information in the above-mentioned text fields should be emptied.
- When a new *Produce* or *PreparedMeal* object was created, the information about this new item should be displayed at the end of a list in the text area. The format of the output for each item should be identical to the formatting in the *toString()* method for both item types (how convenient 😊 )
- If a new item was added, the displayed values for the total price, total tax and total amount (price + tax) should be updated.

#### *Optional:*

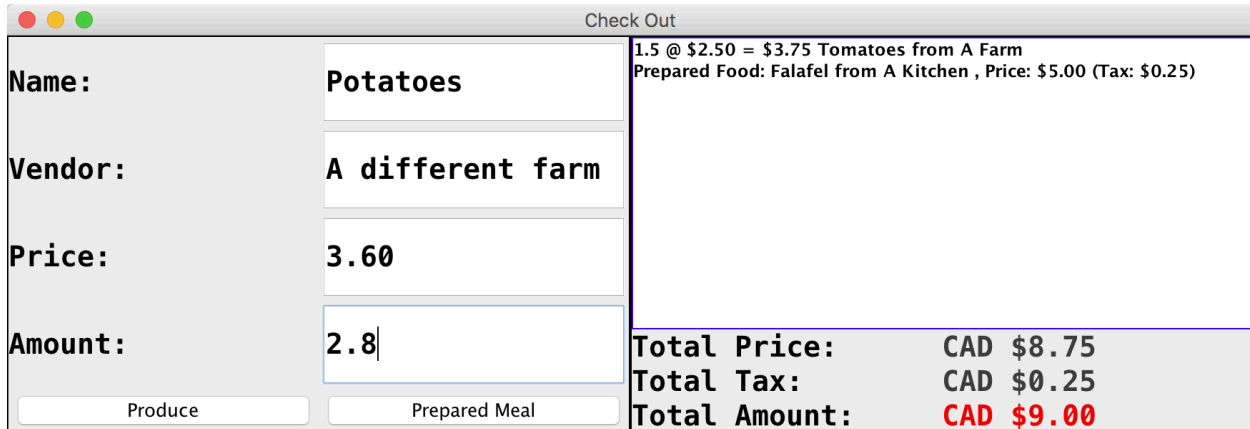
If you want, you can extend your program with additional features, instructions for the user, sound effects, your name and logo in a corner, etc. However, any of these additional features will not count toward your mark for this assignment.

#### *Error checks:*

You may assume that the user types in numerical values in the price and amount fields. However, please note that the amount value for *Produce* is from type double and the amount value for the *PreparedMeal* is from type int. The user might use a floating-point value for the amount in both cases. You can handle this situation by just rounding to the next integer value and use the rounded value as the amount for a *PreparedMeal* item.

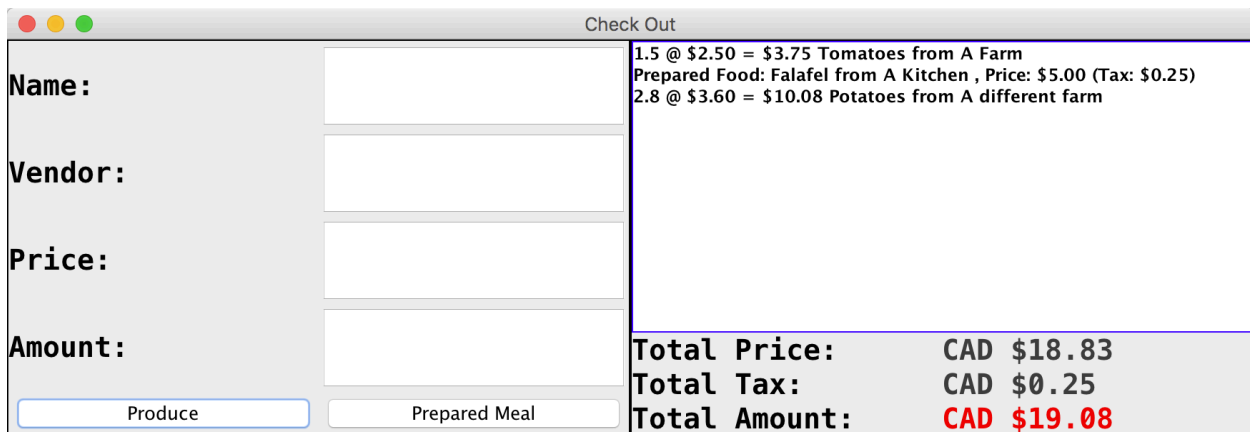
You are welcome (and encouraged) to design your own interface. Be creative. You can add icons, colors, instructions, etc.

If you have trouble getting started with your design or with the understanding of the requirements, I have attached two snapshots of a possible design. But remember, your program **does not have to look like this**:



| Check Out   |                   |
|---|-------------------|
| <b>Name:</b>  | Potatoes          |
| <b>Vendor:</b>  | A different farm  |
| <b>Price:</b>   | 3.60              |
| <b>Amount:</b>  | 2.8               |
| <input type="button" value="Produce"/> <input type="button" value="Prepared Meal"/>                               |                   |
| 1.5 @ \$2.50 = \$3.75 Tomatoes from A Farm<br>Prepared Food: Falafel from A Kitchen , Price: \$5.00 (Tax: \$0.25) |                   |
| <b>Total Price:</b>   | CAD \$8.75        |
| <b>Total Tax:</b>   | CAD \$0.25        |
| <b>Total Amount:</b>  | <b>CAD \$9.00</b> |

Figure 1: Example Checkout program. Snapshot while the user fills in the information for a new food item.



| Check Out  |                    |
|--|--------------------|
| <b>Name:</b>   |                    |
| <b>Vendor:</b>   |                    |
| <b>Price:</b>  |                    |
| <b>Amount:</b>   |                    |
| <input type="button" value="Produce"/> <input type="button" value="Prepared Meal"/>  |                    |
| 1.5 @ \$2.50 = \$3.75 Tomatoes from A Farm<br>Prepared Food: Falafel from A Kitchen , Price: \$5.00 (Tax: \$0.25)<br>2.8 @ \$3.60 = \$10.08 Potatoes from A different farm |                    |
| <b>Total Price:</b>  | CAD \$18.83        |
| <b>Total Tax:</b>  | CAD \$0.25         |
| <b>Total Amount:</b>   | <b>CAD \$19.08</b> |

Figure 2: Example Checkout program. Snapshot after the user clicked the Produce button.

Some tips:

- All Wrapper classes for the numerical primitive datatypes (Integer, Double, etc.) have a method to parse a String (parseInt(String s), or parseDouble(String s)). This might be useful in your implementation.

## 2. Project: Pico-Fermi-Bagel Game

In 2016 Ontario Ministry of Education renewed their commitment to help students gain the math knowledge and skills they will need for the future. One key element in this strategy is to include a play-based model for math learning during the early years.

Your task is to implement a simple math game for these young learners. Your target group will be kids between the age of 5-8. The game is a well-known number guessing game: “Pico, Fermi, Bagel”.

### How the game is played?

The game begins with the computer secretly choosing a random number **with no repeating digits**. For our implementation, we will choose only two-digit numbers. The player will attempt to guess the digits of the number. The computer will respond to each guess with:

- “Bagel”: if the guessed digit is not part of the number
- “Pico”: if the digit is in the number, but in the wrong spot.
- “Fermi”: if the digit is correct (both the value and position is correct).

### For example:

Let’s say the computer randomly chose 37 as the secret number. The player is guessing numbers in the following order:

Guess 1: 12

Computer responds: Bagel Bagel (neither of the two digits are in the number)

Guess 2: 53

Computer responds: Bagel Pico (5 is not in the number; 3 is a digit of the number but in the wrong place)

Guess 3: 39

Computer responds: Fermi Bagel (3 is correct; 9 is not in the number)

Guess 4: 37

Computer responds: Fermi Fermi (3 is correct; 7 is correct).

OK, that is a lot of fun for the kid and it introduces young kids to numbers between 10 and 99. However, it might be hard for them to remember which numbers were already checked. Also, the real educational value comes when the kids can “visualize” the relationship between the two digits of the number. Therefore, your game should have a graphical representation of a number chart, similar to the one in Figure 1:

|    |    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|----|
| 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
| 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |
| 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 |
| 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 |
| 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 |
| 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 |
| 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 |
| 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 |

Figure 1: Number chart

When a “Bagel” digit was guessed, the corresponding row and column in the number chart which contains the incorrect digit should somehow be marked as “cannot be the solution.”

In our example: After the Guess 1 (12) all numbers in the 1. and 2. rows, as well as all numbers in the 2. and 3. columns cannot be the secret number (since both digits got a “Bagel”). Figure 2 shows an example how such markings might look.

|    |    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|----|
| 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
| 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |
| 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 |
| 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 |
| 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 |
| 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 |
| 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 |
| 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 |

Figure 2: Areas are marked as “cannot be the solution.”

In Figure 2, the numbers which cannot be part of the solution are marked with a different background color. You may choose a different way of marking them (change of text color, removing the numbers, maybe even replacing the numbers with a bagel icon, etc.).

Your task is to design and implement the game. The requirements for your program are:

- The game should have a GUI. All user interactions should be part of the GUI.
- Your program should follow the rules of the game (as described above).
- Your game should have a number chart which keeps track of which numbers are not possible solutions.



- Your program should allow the player to enter two single digits or one double-digit number. The input may happen via keyboard (the player types the numbers) or you may have number buttons which the player can select instead.
- Your program should allow the player to check the current guess. This may happen by clicking a button.
- Your program should give feedback regarding the current guess, such as Bagel, Pico or Fermi for each digit. You may also choose to use icons instead, or the letters 'B', 'P', or 'F'.
- If one of the digits is correct ("Fermi"), the player should no longer be allowed to edit this digit in future guesses.
- Your program should count the number of tries it took for the player to guess the secret number.
- Your program should acknowledge when the player has successfully guessed the number. This could be a celebratory message, or 'confetti' or a 'thumbs up' icon, or ...
- Your program should have an "Again" option, which should reset all GUI components and the computer chooses a new random number. So the kid can play again, and again, and again, ...
- When the computer chooses a new random two-digit number, you must print the number in the console (using a `System.out.println` statement). OK, I know that printing out the "secret" number may seem to be against the idea of the game. However, it's necessary to make it possible for the TA's to test and mark your game. It will also make it easier for you to test and debug your program.  
In the future, if you want to distribute the game to younger siblings, nieces, nephews, ... just take the print statement out.

#### Optional:

If you want, you can extend your game with additional features such as instructions for the user, sound effects, your name and logo in a corner of the game, etc. However, any of these additional features will not count toward your mark for this assignment.

#### Error checks:

Your players will be very young kids and **MANY** things could go wrong. However, for this assignment you may assume that you have only very well-instructed and well-behaved kids as users who will only type in a single digit in a spot where single digits are required, etc. No error checks required.

If you plan to distribute the game later to friends and family, you might need to add some type checks and exceptions before doing so.

How your interface is designed and how your program is structured is up to you. Here are some tips on how to approach the assignment:

- Familiarize yourself with the game by playing it first with somebody using a piece of paper and a pencil. Play a few rounds. You may also use a number chart on the side.

- Make a design for the GUI on paper.
- Make a program design on paper (UML diagram). Although it is not required for this assignment, I recommend not to have all the code in the main method. Instead identify parts of the program which can be encapsulated, such as the number chart. In the next step, identify the required communication between the classes (methods to invoke). The game is easy enough that your class structure will be very simple - no complicated inheritance or abstract classes, but it offers you a perfect opportunity to practise your understanding of object-oriented programming. Furthermore, if you work as a team, a separation of the program into different classes allows you to split the work between the team members.

*Note: You do not need to submit any documentation of this design step. Your program will speak for itself.*

#### **Some programming tips:**

- Note: For the implementation, it might be easier to manage the “secret” two-digit number as two single-digit numbers.
- All Wrapper classes for the numerical primitive datatypes (Integer, Double, etc.) have a method to parse a String (parseInt(String s), or parseDouble(String s)). This might be useful in your implementation.
- Remember that Swing components are all derived classes from the java.lang.Object class. Therefore, you can “organize” reference variables for GUI objects in an Java array. Such as

```
JPanel[][] panel_array = new JPanel[9][10];
```

Of course this works also for other GUI components, not just for JPanel. Keep in mind that the array is only managing the reference variables for a GUI component and has nothing to do with the layout manager. Managing the layout components in a container is a totally different problem. For more information, you might want to check out the lecture code from March 24<sup>th</sup>.

- To avoid users from making any further editing to a GUI component (such as text field), you can set the editable flag to false.