Code Assignment 1
Graphic User Interface & Methods

CS152 – Computer Programming Fundamentals

Instructor: M. Wolverton

Items Due

- **Completed Code 1**

Instructions

Complete the coding directive in an Intellij IDEA project with the assignment name (e.g. Code 1, Code 2-3 etc.) and package name cs152. Compress each completed project separately into a .zip file and upload it to your account on the file server at cec-code-lab.aps.edu in a folder named CS152. Code will be downloaded and archived for grading on the assignment due date.

Directive

Rebuild any one of the assignment numbered 3-8 from the first semester – but this time with a JavaFX based Graphic User Interface (GUI). Please satisfy all technical critera below. The original documents for assignments 3-8 are attached for reference on following pages as an appendix.

Code 1

- Your program should contain all of the **core functionality** of the command line program you chose – except for descibed print and display functionality. All output and user input should be handled through the GUI. Optional challenge features are not required for credit but are encouraged.

- Your program should make use of input **event handling** of button action, mouse, or keyboard events to time code execution.

- Create at least **one extra method** (beyond `main()` and `start()`) that is called during run-time of the program.

- Use **at least two layout Nodes** from the following list:

`Pane`	`StackPane`	`GridPane`
`HBox`	`Vbox`	`AnchorPane`
`BorderPane`	`TilePane`	

- Your Scene layout should contain **at least two layout nodes that are 'nested'**. Nested layouts are child nodes of another layout. For instance your outermost layout node could be a Vbox to create a vertical column, which then has a child Hbox to hold two buttons side by side.

Appendix

Semester 1 Assignments

List of Eligible Assignments

Assignment 3: Calculator

Assignment 4: Rock-Paper-Scissors

Assignment 5: Higher-Lower Game

Assignment 6: Dice Simulation

Assignment 7: Tic-Tac-Toe

Assignment 8: MineSweeper

Original assignment documents are on the following pages.

Assignment 3
While Loop

CEC – Computer Science
Instructor: M. Wolverton

Items Due

- **Source Code for Menu-Calculator**

Instructions

Complete the coding directive in an Intellij IDEA project with the assignment name (e.g. Menu-Calculator) and package name ceccs. Login to the file server at cec-code-lab.aps.edu and create a folder with the assignment name (e.g. Assignment 3). Locate the `fileName.java` (e.g. `Main.java`) files of your source code, then upload them to into the assignment folder. Code will be downloaded and archived for grading on the assignment due date.

Be careful to look at the specific requirements for each program!

Menu-Calculator

Write a calculator program that uses the text terminal and repeats until the user selects a quit option. The user selects an operation, then gives two inputs. Then reports the result. After each use, the user is presented with same menu until they select a quit option.

- Print program title or welcome message identifying the program and what it is.
- Present a menu of options. Options can be input as either numbers, characters or words. The menu options should include
  *add *subtract *multiply *divide *quit
- If the user selects something besides quit, prompt the user for both values. Store them as double type.
- Create a String variable to hold the output message.
- Print the output message.
- Repeat the menu until quit is selected.
- Detect invalid menu selections, and print an error message followed by the menu again.

Sample Output

```
This program is a four-function calculator.
What would you like to do?
[1] add    [2] subtract    [3] multiply    [4] divide    [0] quit
3
Input the first value.
25
Input the Second value.
-.5
25.0 * -0.5 = -12.5
What would you like to do?
[1] add    [2] subtract    [3] multiply    [4] divide    [0] quit
0
Goodbye.
```

Assignment 4
Pseudo-Random Numbers

CEC – Computer Science
Instructor: M. Wolverton

Items Due

- **Source Code for Rock-Paper-Scissors**

Instructions

Complete the coding directive in an Intellij IDEA project with the assignment name (e.g. Rock-Paper-Scissors) and package name ceccs. Login to the file server at cec-code-lab.aps.edu and create a folder with the assignment name (e.g. Assignment 4). Locate the `fileName.java` (e.g. `Main.java`) files of your source code, then upload them to into the assignment folder. Code will be downloaded and archived for grading on the assignment due date.

Be careful to look at the specific requirements for each program!

Rock-Paper-Scissors

Write a command line rock-paper-scissors game where the player is versus the computer. Gameplay should be menu or text command based, and should repeat until the player issues a quit command. The game should also keep track of score.

- The player may make choices with either of two options:
  - A number or letter based menu (e.g. [1] Rock … [0] Quit)
  - A text command based system (e.g. player types “paper” or “quit”)
- Random Number Generation should be used to make the computer player’s selection.
- The program should output a message after each round confirming what selections were made by each player, and indicating which player won or if a draw occurred.
- After each round, the current score statistics should be shown:
  - Human player’s wins
  - Computer player’s wins
  - Draw rounds.

Sample Output

```
----------------------------
-----Rock Paper Scissors-----
----------------------------

Choose your weapon!
[r] Rock    [p] Paper    [s] Scissors   [q] quit
s
You have chosen Scissors while the computer has chosen Paper.
Scissors cuts paper. You win!
Score –      Player: 1    Computer:0   Draw:0

Choose your weapon!
[r] Rock    [p] Paper    [s] Scissors   [q] quit
q
Goodbye.
```

Optional Challenges

- Difficulty Settings: Create a ‘difficulty’ setting which makes it *slightly* more or less probable for the computer to win.
- Player Select: Create an option for two human players to play.
- Random Play for Human: Create a random option for the human player each turn.

Assignment 5
Flow Control & State Variables

CEC – Computer Science
Instructor: M. Wolverton

Items Due

- **Source Code for Higher-Lower-Game**

Instructions

Complete the coding directive in an Intellij IDEA project with the assignment name (e.g. Higher-Lower-Game) and package name ceccs. Login to the file server at cec-code-lab.aps.edu and create a folder with the assignment name (e.g. Assignment 5). Locate the `fileName.java` (e.g. `Main.java`) files of your source code, then upload them to into the assignment folder. Code will be downloaded and archived for grading on the assignment due date.

Be careful to look at the specific requirements for each program!

Higher-Lower-Game

Write a command line game where the player is supposed to guess a random number between 1 (inclusive) and 1000 (exclusive). After each guess, they are told if their answer is correct, and whether they guessed too high or too low. The game is over after a fixed number of turns (e.g. 5) or they have guessed the value.

After each round they should be presented with a score board and an option to play again.

Additional criteria:

- The winning number, and player's guess should be stored as an `int`.
- After each turn, the program should remind the player how many guesses they have remaining.
  - At 1 guess, the hint text should not be plural.
- The score board should contain wins and losses.
- If a guess is invalid (out of range) print an error message and do not consume a turn.

Sample Output

```
-----------------------------------
-----Higher-Lower Guessing Game-----
-----------------------------------
Guess a number between 1 and 1000. You have 5 Guesses Remaining.
400
Too High!
 ...
Guess a number between 1 and 1000. You have 3 Guesses Remaining.
100
Too Low!
 ...
Guess a number between 1 and 1000. You have 1 Guess Remaining.
175
Too High!
You Lose. So sorry.
Wins: 0 Losses: 1
Do you want to play again? (y/n)
n
Goodbye.
```

Optional Challenges

- Difficulty Settings: Create a 'difficulty' options menu. Easier would involve a smaller range (e.g. 1 … 100) and more guesses, Harder would involve a larger range and fewer guesses.
- Guess List: After every turn, have the program list all previous guesses along with their hints (too high, too low).
- Reverse Play Mode: A different game mode where the user thinks of a number and the computer generates guesses. The player is given 3 options every guess e.g. `[1] guess is too low  [2] you got it! [3] guess is too high`

Assignment 6
Static Methods & Array Basics

CEC – Computer Science
Instructor: M. Wolverton

Items Due

- **Source Code for Dice-Stats**

Instructions

Complete the coding directive in an Intellij IDEA project with the assignment name (e.g. Dice-Stats) and package name ceccs. Login to the file server at cec-code-lab.aps.edu and create a folder with the assignment name (e.g. Assignment 6). Locate the `fileName.java` (e.g. `Main.java`) files of your source code, then upload them to into the assignment folder. Code will be downloaded and archived for grading on the assignment due date.

Be careful to look at the specific requirements for each program!

Dice-Stats

Write a command line program that simulates statistics for rolls of six-sided dice pairs. The program should simulate a requested number of random dice rolls, then report back the total number of results for each value rolled (e.g. 2's rolled, 3's rolled … 12's rolled).

Additional criteria:

- Create a new **static method** to handle a single die roll that has the following declaration:

```
static int rollDie() {
       // logic to return a number 1 – 6 with equal probability
}
```

- The program should ask the user how many trials to run (total rolls).
- Create and use an int **array** keep a total count for each outcome (2's rolled, 3's rolled … 12's rolled).
- Print the total number of each roll.
- Print the calculated probability of each roll based on the simulated run.

Sample Output

```
-------------------------------------------
-----Six-Sided Dice Pair Roll Statistics-----
-------------------------------------------
How many times to roll dice?
1000
Simulating Rolls…
Here are the number of times each roll occurred:
 2: 28
 3: 56
 4: 110
       ...
 12: 30
Here are the probabilities for each roll during this trial:
 2: 2.8%
 3: 5.6%
 4: 11.0%
       ...
 12: 3.0%
```

Optional Challenges

- Make the program ask for how many dice to roll. Be sure to account for all roll possibilities.
- Make the program ask how many sides each die has. Be sure to account for all roll possibilities.

Assignment 7
2D Arrays & For Loops

CEC – Computer Science
Instructor: M. Wolverton

Items Due

- **Source Code for Tic-Tac-Toe**

Instructions

Complete the coding directive in an Intellij IDEA project with the assignment name (e.g. Tic-Tac-Toe) and package name ceccs. Login to the file server at cec-code-lab.aps.edu and create a folder with the assignment name (e.g. Assignment 7). Locate the `fileName.java` (e.g. `Main.java`) files of your source code, then upload them to into the assignment folder. Code will be downloaded and archived for grading on the assignment due date.

Be careful to look at the specific requirements for each program!

Tic-Tac-Toe

Write a command line version of the game Tic-Tac-Toe. Leverage methods, 2d arrays and for-loops to help organize your code. Program the game with the intent that it will be played by two human players. A computer player may be optionally added later.

Additional criteria:

- Have the players use the keyboard numpad coordinates to select a play location.
- Game data should be stored in a 2d char array.
- Use a pair of nested for( ) loops for all operations that need to access the whole game data array. For instance initialization.
- Create a method to print the game board with the following returns and parameters:

```
static void printGameGrid(char[3][3] gameData) {
       // logic to print out the game board in a visually appealing way
}
```

- Create a method to check for win and draw states. It should return a char with the following return codes:

```
static char checkWin(char[3][3] gameData) {
       // logic to look for win states or draw.
       // returns ‘X’ if X wins, ‘O’ if O wins, ‘D’ if draw and ‘N’ for none.
}
```

- Ask players If they want to keep playing at the end of each round
- X plays first, but the losing player should always play X’s the following round. For a draw, players switch their marks.
- Print the score count (P1 wins, P2 wins, Draws) at the end of each round.

Sample Output

```
[[ Tic-Tac-Toe ]]
Player 1 (X): use Numpad (1-9) to select a location.
         - | - | -
        -----------
         - | - | -
        -----------
         - | - | -
3
         - | - | -
        -----------
         - | - | -
        -----------
         - | - | X
...
```

Optional Challenges

- Create a computer opponent that makes random moves.
- Create a computer opponent that makes strategic moves.
- Create a difficulty setting menu (easy, normal, hard).

Assignment 8
Object Orientation

CEC – Computer Science
Instructor: M. Wolverton

Items Due

- **MineSweeper.java – Program Class Source Code**
- **Grid.java – Grid Class Source Code**
- **Tile.java Tile Class Source Code**

Instructions

Complete the coding directive in an Intellij IDEA project with the assignment name (e.g. Minesweeper) and package name ceccs. Login to the file server at cec-code-lab.aps.edu and create a folder with the assignment name (e.g. Assignment 8). Locate the `fileName.java` (e.g. ClassName`.java`) files of your source code, then upload them to into the assignment folder. Code will be downloaded and archived for grading on the assignment due date.
Be careful to look at the specific requirements for each program!

MineSweeper

Write a command line version of the game MineSweeper. Create at least 3 classes and leverage object orientation by creating two classes designed for instantiation and one that contains main as the program itself. Also make use of methods, 2d arrays and for-loops to help organize your code. The game should be designed for replayablilty with a totally random minefield for each play.

The following classes should be in your code, with the following methods and fields. You may add extra methods and fields.

Tile Class

Fields:

booleans: `isCovered  isBomb  isFlagged` int: `numAdjBombs`

Methods:

`Tile()` - A constructor function to initialize all variables
`void toggleFlag()` - A function to toggle isFlagged from true to false.

Grid Class

Fields:

ints: `width  height` `Tile[][] tiles`

Methods:

`Grid(int w, int h)` - A constructor function which sizes the array tiles and initializes all tiles in it.
`void printGrid()` - Displays the game map using text symbols on the System.out console.
`void createRandomMinefield(int numMines)` - Randomly places the desired number of mines.
`void calcAdjNums()` - Calculates the number of bombs surrounding each tile and stores it in numAdjBombs.
`boolean checkLoss()` - A check for any uncovered bomb to check if player has lost the game.
`boolean checkWin()` - Checks that no ‘empty’ tiles are covered to check if player has win the game.

MineSweeper Class

Methods:

`public static void main(String[] args)` - The start execution of the program.

Additional criteria

- The game must detect the loss condition that any uncovered bomb represents a loss.
- The game must detect the win condition that all covered tiles contain bombs, and no ‘empty’ tiles are covered.
- The game should randomly distribute bombs differently every time the game is played.
- Print the entire game field every turn with symbols for covered tiles, uncovered bombs, 0-adjacency uncovered tiles, and adjacency covered tiles.
- The player should select tiles with a row and column based system.
- Adjacency numbers that represent the number of nearby bombs must be accurate for all tiles and all map possibilities.
- Gameplay should be as free from crashes as possible – it takes time to play minesweeper and crashes are frustrating.

Optional Challenges

- Make it impossible to lose on the first turn. Hint: configure the minefield after the first play coordinates are selected.
- Make selections automatically select all tiles surrounding any tile with zero adjacent bombs – and continues selecting until all tiles surrounding successively uncovered zeros are selected.
- Add the ability to flag uncovered tiles along with a symbol for flagged tiles in the printout.
- Make it so that no bombs are placed within 1 tile of the first selection tile (a 9 tile ‘safe’ square).
- Add difficulty settings that pre-configure different sizes of map with different counts of bombs.