

Project 3: Stick, Water, Fire

Using conditionals and the Random class.

Overview	1
Learning Goals:	1
Program Interaction.	2
Program Development- Your Tasks.	3
Testing	5
Export and Submit	6
Style Guide:	8

Overview

For this project, you will develop a game program called Stick, Water, Fire- a version of the game Rock, Paper, Scissors. The user plays against the computer. You will use conditional expressions in several ways in this project. Primarily, you will write code to implement the game rules using conditional statements. You will also utilize Java's `Random` class to generate random outcomes used in the game for the computer's choice. The `Random` class can be seeded so that development and testing can be more easily done. Another aspect of this project is the use of a private method in a class. Private methods are used only within a class to help other methods with their calculations.

Learning Goals:

- Work with multi-file Java projects.
- Write instance variables and methods.
- Use a private method in a class.
- Work with conditionals.
- Work with the `Random` class.
- Learn and use `String` methods
- Practice testing and using the JGrasp debugger.

Program Interaction.

Our goal is to make a fun and interactive game for users. The user will be able to enter their choice of Stick, Water, or Fire. The computer will also make a choice, and the winner will be chosen based on the following three rules:

1. Stick beats Water by floating on top of it
2. Water beats Fire by putting it out
3. Fire beats Stick by burning it

The user is prompted to enter S, W or F (Note that the input can be upper or lower case). The computer then makes its choice (randomly generated) and the program applies the game rules to the user and computer's choices.

The user and computer's scores are recorded as well as the number of rounds played.

Ties result in no increase in either user or computer scores, but the round is counted.

A sample of the program's behavior is given below. This is how the program should function when you have it working:

```
Welcome to Stick-Water-Fire!

Rules:
  You will play against the computer for the specified number of rounds.
  You will make a choice: 'S', 'W', or 'F' to represent 'Stick', 'Water', or 'Fire'.
  The computer will also make a choice, and the winner is chosen as follows:
    Stick beats Water (it floats on top)
    Water beats Fire (extinguishes the fire)
    Fire beats Stick (burns the stick)
  In the event of a tie, there is no winner.
  Each round, the winner will have their score incremented.
  A report of scores and number of rounds will be displayed at the end of each round.
  Enter X to quit.
  Good luck!
Enter 'S' for Stick, 'W' for Water, 'F' for Fire, or X to quit:
```

Now an example of user input to the program:

```

Enter 'S' for Stick, 'W' for Water, 'F' for Fire, or X to quit:
S
You chose S and the computer chose F
You lost. Better luck next time!

Game summary:
Total plays: 1
Your total score: 0, computer total score: 1

Enter 'S' for Stick, 'W' for Water, 'F' for Fire, or X to quit:
W
You chose W and the computer chose F
You won! Nice job!

Game summary:
Total plays: 2
Your total score: 1, computer total score: 1

Enter 'S' for Stick, 'W' for Water, 'F' for Fire, or X to quit:
F
You chose F and the computer chose F
You tied!
Game summary:
Total plays: 3
Your total score: 1, computer total score: 1

Enter 'S' for Stick, 'W' for Water, 'F' for Fire, or X to quit:
X
Thanks for playing!

```

There is also a little input validation in this program:

```

Enter 'S' for Stick, 'W' for Water, 'F' for Fire, or X to quit:
z
    Invalid input entered: z

Enter 'S' for Stick, 'W' for Water, 'F' for Fire, or X to quit:

```

Note that on an invalid user input, the computer wins and a round is tallied (humans make mistakes but computers do not!).

Program Development- Your Tasks.

The Starter Code consists of two classes: `StickWaterFireMain` and `StickWaterFireGame`.

The `StickWaterFireMain` class carries out the interaction between the player and the instance of the `StickWaterFire` game object. ***Do not change this file.*** The `StickWaterFireGame` class keeps the state of the game, plays rounds of the game, and provides a report of the number of wins for the player and computer, and the total number of rounds played.

You will develop code in the `StickWaterFireGame` class only. Your tasks are indicated by a `TODO`. You will also find more instructions in the form of comments that detail more specifically

what each part of the class is supposed to do, so make sure to read them. ***Do not delete or modify the method headers in the starter code!***

Here is an overview of the TODOs in the class.

TODO 1: Declare the instance variables of the class. Instance variables are private variables that keep the state of the game. The recommended instance variables are:

1. A variable, "rand" that is the instance of the Random class. It will be initialized in a constructor (either seeded or unseeded) and will be used by the getRandomChoice() method.
2. A variable to keep the player's cumulative score. This is the count of the number of times the player (the user) has won a round. Initial value: 0.
3. A variable to keep the computer's cumulative score. This is the count of the number of times the computer has won a round. Initial value: 0.
4. A variable to keep the count of the number of rounds that have been played. Initial value: 0.
5. A variable to keep track of if the player has won the current round (round-specific): true or false. This would be determined by the playRound method. Initial value: false.
6. A variable to keep track of if the player and computer have tied in the current round (round-specific): true or false. This would be determined by the playRound method. Initial value: false.

The list above contains the minimum variables to make a working program. You may declare more instance variables as you wish.

TODOs 2 and 3: Implement the constructors. The constructors assign the instance variables to their initial values. In the case of the Random class instance variable, it is initialized to a new instance of the Random class in the constructors. If a constructor has a seed parameter, the seed is passed to the Random constructor. If the constructor does not have a seed parameter, the default (no parameter) Random constructor is called.

TODO 4: This method returns true if the inputStr passed in is one of the following: "S", "s", "W", "w", "F", "f", false otherwise. Note that the input can be upper or lower case.

TODOs 5, 6, 7, 8, 9, 10 These methods just return the values of their respective instance variables.

TODO 11: This is a private method that is called by the playRound method. It uses the instance variable of the Random class to generate an integer that can be "mapped" to one of the three Strings: "S", "W", "F", and then returns that String, which represents the computer's choice.

TODO 12: The playRound method carries out a single round of play of the SWF game. This is the major, "high-level" method in the class. This method does many tasks including the following steps:

1. Reset the variables that keep round-specific data to their initial values.
2. Assign the computer's choice to be the output of the getRandomChoice method.
3. Check that the player's choice is valid by calling isValidInput. If the player's input is not valid, the computer wins by default. This counts as a round, so the number of rounds is incremented.
4. If the player's choice is valid, the computer's choice is compared to the player's choice in a series of conditional statements to determine if there is a tie, or who wins this round of play according to the rules of the game:

S beats W

W beats F

F beats S

Both have the same choice- a tie.

The player and computer scores are updated depending on who wins. In the event of a tie, neither player has their score updated. Finally, the number of rounds of play is incremented.

Note: Do not duplicate the isValidInput or the getRandomChoice code in playRound. Call these methods from playRound. The point is that delegating tasks to other methods makes the code easier to read, modify and debug.

Testing

Test your code on your machine before submitting to Gradescope. If your code does not work correctly on your computer it will not pass many of the tests on Gradescope. jGrasp is designed to develop code, and so that is the tool you should use to test and develop your project code.

Seeding the Random Number Generator

To begin with, you want to use the seeded version of Random. You can select the version by commenting out one of the constructor calls. This image shows the seeded version is selected:

```
7 //StickWaterFireGame game = new StickWaterFireGame();  
8 StickWaterFireGame game = new StickWaterFireGame(1234);
```

You can then test your code as you know the sequence of computer choices will be the same each time you run the program. you can also try different seed values. When you are confident your program is working correctly, switch to the unseeded version and continue to test.

Test Cases

Ideally, you want to test all possible combinations of player and computer choices against the outcome determined by the logic of your playRound method. The number of possible


combinations is $3 * 3 = 9$. You also want to test an invalid input. This table summarizes the test cases and expected outcomes:

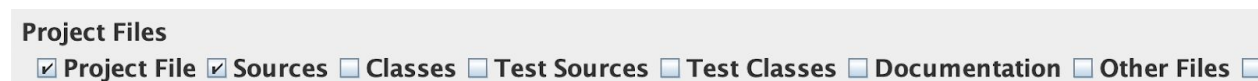
Player	Computer	Winner
S	S	Tie
W	S	Computer
F	S	Player
S	W	Player
W	W	Tie
F	W	Computer
S	F	Computer
W	F	Player
F	F	Tie
invalid		Computer

Export and Submit

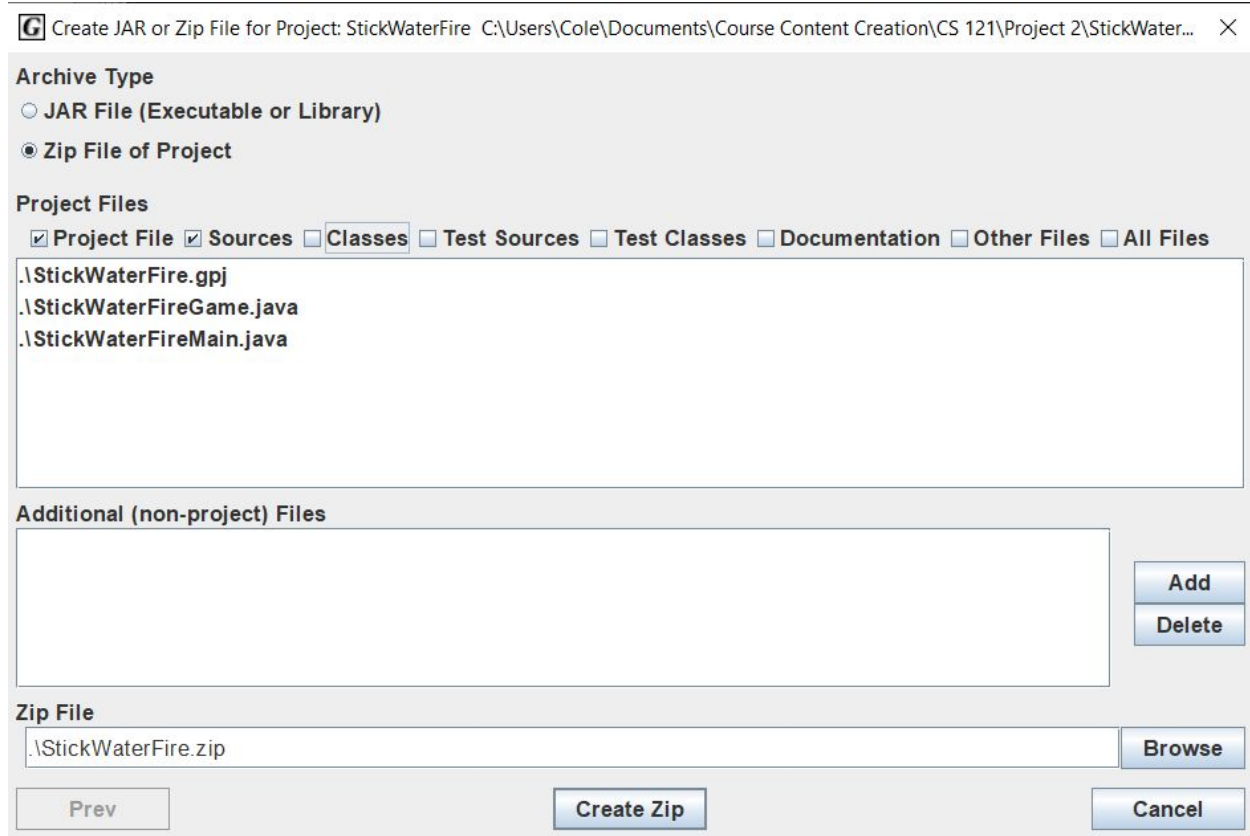
When you have completed this project, you need to create a zip file that contains the entire Java project. This is done in jGRASP in the following manner.

1: Create the zip file.

Click on the “jar”  icon. Select **Project File** and **Sources**.



Click **Create Zip** and the .zip file is created. See the figure below.



Note: We require that you create a `zip` file to upload to Gradescope.

Important Note: If you are not using **jGrasp** to develop your code, you must ensure that your zip file has the correct file structure. Contact the course staff if you have any questions about the correct structure of the project zip files.

Step 2: Submit the file in Gradescope

Now log into Gradescope, select the assignment, and submit the zip file for grading.

Compilation errors will be provided by the autograder for this assignment. You must read the assignment carefully -- if your submission is not passing a test, it is most likely because your submission doesn't match the requirements of the assignment.

Remember, you can re-submit the assignment as many times as you want, until the deadline. If it turns out you missed something and your code doesn't pass 100% of the tests, you can keep working until it does. Attend office hours for help or post your questions in Piazza.

Style Guide:

Apart from the score of the autograder, your project will be inspected for following these style guidelines:

1. All unused lines of code have been removed.
2. Optimal use of blank lines and indents and spaces for operators.
3. All variable names are understandable.

See zyBooks sections 2.8 and 2.14 as well code examples from the course material for the recommended style guidelines.