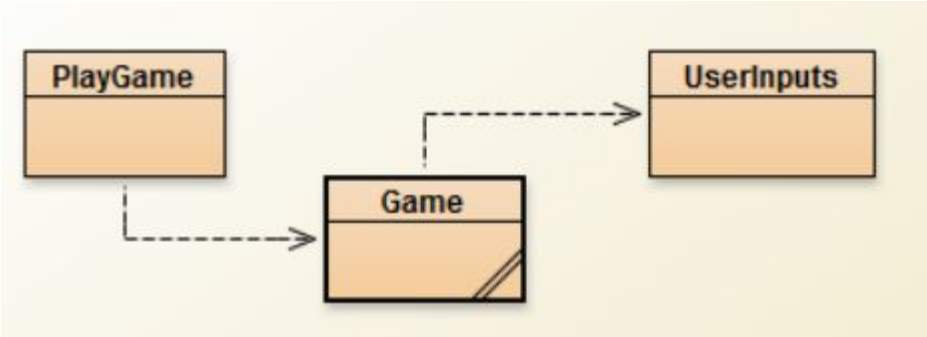
Criterion C: Development

All Classes:



Techniques used:

A. Lists to store multiple items like game wages, menu options and puzzle phrases.
B. HashMap for guessed letters in which we use key and value pair.
C. While loops to have a different output occur
D. If else statements in dealing with certain situations where the data input is incorrect
E. Switch case for the menu options
F. Calling methods from other classes for user input
G. Try and Catch to catch any wrong input besides the options displayed

In this program we implement a phrase guessing game in which a random phrase is picked up from the array of different phrases. The phrase come with empty lines so that the user guess the Character in each phrase. Also, random amounts of cash will be added into the user account. Now the user can select different options in the menu. For first option, the user enters a letter. If the letter is correct, the user will win money and will be added before into user account. If the user's guess is invalid, then the $150 will be deducted from the user's account. If the user faces difficulties in guessing the phrase, then the user has the option to buy vowels for $250. Then, user can choose the appropriate vowel in order to guess the phrase. If the user continuously guesses the wrong letter, then his balance will be deducted for each wrong answer. If the user's balance equals to zero, the game will over. User can also choose to solve the puzzle. At anytime, the user can check their balance or quit the game.

Program File:


Puzzle
Activity

A. Lists

```
// The different puzzles to choose from
private static final List<String> _puzzles = Arrays.asList(
        "THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG",
        "WAR BEH KEH",
        "INFORMATION SCIENCES AND TECHNOLOGY",
        "A DIAMOND IN THE ROUGH",
        "BEATING THE ODDS",
        "HOW ARE YOU",
        "ZOOM IN ZOOM OUT"
    );
```

```
// Choose one of the puzzles at random
String puzzle = chooseRandomPuzzle();

String str = chooseRandomWedgeValue();
```

I implemented a List function into the program. For the puzzle phrases, I created a List<String> _puzzles = Arrays.asList which I can easily call and retrieve. I imported a java.util.Random class to randomly pick a puzzle to be retrieved for the game. I used this so the user will have a random puzzle instead of choosing hidden puzzles.

```
private static final List<String> game_wedges = Arrays.asList(
        "$5000",
        "$600",
        "$500",
        "$300",
        "$500",
        "$800",
        "$550",
        "$400",
        "$300",
        "$900",
        "$500",
        "$300",
        "$900",
        "BANKRUPT",
        "$600",
        "$400",
        "$300",
        "$800",
        "$350",
        "$450",
        "$700",
        "$300",
        "$600"
    );
```

Another instance of the List function can be seen with the cash amount that a user can land on when spinning the wheel, or choosing option 1. Varying amounts are stored in the list and the List can cache the value so the .size() is not called over and over.

B. Hashmap

```
You chose: 1. Spin the wheel
Enter a letter: s
Your letter is: S
Invalid Guess!!!
                    ----------------------
                    *        PUZZLE       *
                    ----------------------


_   _ _ _ _ _ _ _   _ _   _ _ _   _ _ _ _ _

1. Spin the wheel
2. Buy a vowel
3. Solve the puzzle
4. Check your balance
5. Quit the game
Enter choice:
```

```
private static Map<Character, Boolean> guessedLetters = new HashMap<>();
```

```
boolean isLetterGuessed = revealLetters || guessedLetters.containsKey(char_puzzle);
```

```
guessedLetters.put(letter, true);
```

HashMap maintains key and value pairs through HashMap<Key, Value> and implements Map interface. It is used for maintaining key and value mapping. Here it is used for the guessed letters in the puzzle and revealing them. The boolean function returns true or false based on whether the specified key is found in the map. In this case, once a letter has been guessed correctly, it is revealed in the blank spaces printed. I used this method as it is a simple way to reveal the letter to the user if it was correct by entering the correct key.

```
Enter choice: 1
You chose: 1. Spin the wheel
Enter a letter: D
Your letter is: D
You Won: $300
                      ----------------------
                      *        PUZZLE       *
                      ----------------------


_   D _ _ _ _ _ D   _ _   _ _ _   _ _ _ _ _

1. Spin the wheel
2. Buy a vowel
3. Solve the puzzle
4. Check your balance
5. Quit the game
Enter choice:
```

C. While loops

```
public static void gameMenu() {
    // Choice from the menu
    int choice = 0;

    // Line entered from keyboard
    String line = "";

    // True when user wants to quit
    boolean quit = false;
```

```
// Repeat the menu until the user chooses to quit
while (!quit) {
    System.out.println("                    ----------------------");
    System.out.println("                    *        PUZZLE      *");
    System.out.println("                    ----------------------");
    System.out.println("                                        ");

System.out.println(maskPuzzle(puzzle, letters));
System.out.println();
if(win_cash <= 0){
    System.out.println("You don't have enough balance to play!!!");
    System.out.println("***GAME OVER***");
    System.exit(0);

}
// Loop over the menu choices, and display each one
for (String menuChoice : _menuChoices) {
    // Skip blank place-holder choices
    if (!menuChoice.equals("")) {
        System.out.println(menuChoice);
    }
}
System.out.print("Enter choice: ");
line = input.getInput();
try {
    // If the input was not an integer, then that error will be caught
    choice = Integer.parseInt(line);
} catch (NumberFormatException e) {
    // Error message, then go to the top of the loop
    System.out.println("Invalid input");
    win_cash = win_cash -50;
    //System.out.println("Remaining Cash is : $" +win_cash );
    continue;
}

// If not valid, then go back to the top of the loop
if (!isValidMenuChoice(choice)) {
    System.out.println("Not a menu choice");
    continue;
}
```

A while loop statement repeatedly executes a target statement as long as a given condition is true. Here the while loop is used in gameMenu(). It acts as a condition when the user does not want to quit. Otherwise, the game will end and skip the while loop when the boolean quit = false; becomes true. When the condition is active, it will go through the menu choices given by the switch(choice) function. I used this to have a simple choice where the user can quit or the user can continue leading to the various options in the game.

D. If else statements

```
                    ----------------------
                    *        PUZZLE      *
                    ----------------------


_   _ _ _ _ _ _ _   _ _   _ _ _   _ _ _ _ _

1. Spin the wheel
2. Buy a vowel
3. Solve the puzzle
4. Check your balance
5. Quit the game
Enter choice: 1
You chose: 1. Spin the wheel
Enter a letter: A
That's a vowel!
Enter a letter:
```

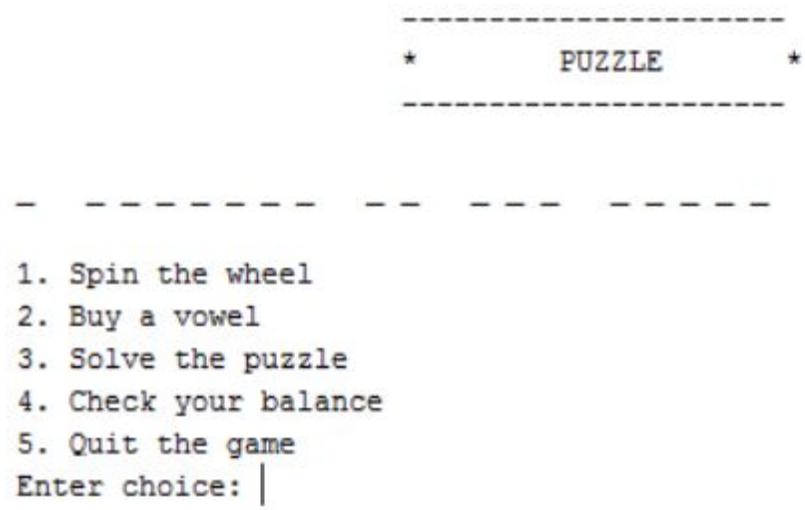
The if/else statement executes a block of code if a specified condition is true. If the condition is false, another block of code can be executed.

```
if(guessedLetters.containsKey(letter)){
    System.out.println("Already Guessed, Enter another Letter");
    finished = false;
}
//Testing for vowel when spinning the wheel
if (inputCase == 1){
    if (letter == 'A' || letter == 'E' || letter == 'I' || letter == 'O' || letter == 'U') {
        System.out.println("That's a vowel!");
        finished = false;
    }
}
```

```
if (inputCase == 2){
    if (letter == 'A' || letter == 'E' || letter == 'I' || letter == 'O' || letter == 'U') {

    } else {
        System.out.println("That's not a vowel!");
        finished = false;
    }
}
```

An If/else statement is implemented here based on the conditions of having a vowel or consonant. For Case 1, previously mentioned with the switch case function, it is used for guessing consonants. An If statement is used to test for the condition of having any vowels being entered and to reject the input. The same If statement is used for case 2 where the user is buying a vowel. The statement is testing for the input to have a vowel, else, the input is an error until a vowel is inputted. This is an easy way to check whether or not a vowel or consonant has been entered for both case conditions.

E. Switch case



All the various options to select in the game. I used a switch case statement to have multiple options so it can be easily tested for equality against a list of values. Each value is a case, and the variable being switched on is checked for each case. This allows it to be easier to implement options than If/else statements.

```
System.out.println("You chose: " + _menuChoices.get(choice - 1));
switch (choice) {
    case _quitChoiceNumber:
    // This will allow us to leave the menu loop
    quit = true;
    break;
```

```
case 2:
char vowel = inputLetter(2);
if(win_cash < vowel_Cash){
    System.out.println("You don't have enough money to buy Vowel");
    break;
} else if(win_cash >=vowel_Cash ){
    System.out.println("Enter vowel: " + vowel);
    guessedLetters.put(vowel, true);
    win_cash = win_cash - vowel_Cash;
    //System.out.println("Your Cash is : $" + win_cash);
    break;
}
break;
```

This is an example of one of the cases implemented in the switch case statements. Case 2 is the second option in the game for buying vowels. It creates a case on what to do when the option is selected and breaks to end the case. The switch case allows the user to select one of the options to be selected. I used it as it made it easy to create different options. The user only has to select one option and the switch case will go through and break once it finished. This was a more efficient way than using multiple If/else statements for creating different menu options.

F. Calling methods from other classes

```
import java.util.Scanner;

//class used to get user input
public class UserInputs {

    Scanner scan = new Scanner(System.in);

    //get string from user
    public String getInput(){
        String line = "";
        line = scan.nextLine();

        return line;
    }
}
```

The UserInputs class calls the gameMenu() method from the Game class. This was done to get user input using the java.util.Scanner class. This is for getting the user to enter a value and interact with the game.

```
public class PlayGame {
    //main method
    public static void main(String[] args) {

        //Object of Game class
        Game game = new Game();
        game.gameMenu();//call method
```

The PlayGame class calls the gameMenu() method from the Game class. The PlayGame class is the main method responsible for running the entire program.

```
public static void gameMenu() {
    // Choice from the menu
    int choice = 0;

    // Line entered from keyboard
    String line = "";

    // True when user wants to quit
    boolean quit = false;
```

G. Try and catch

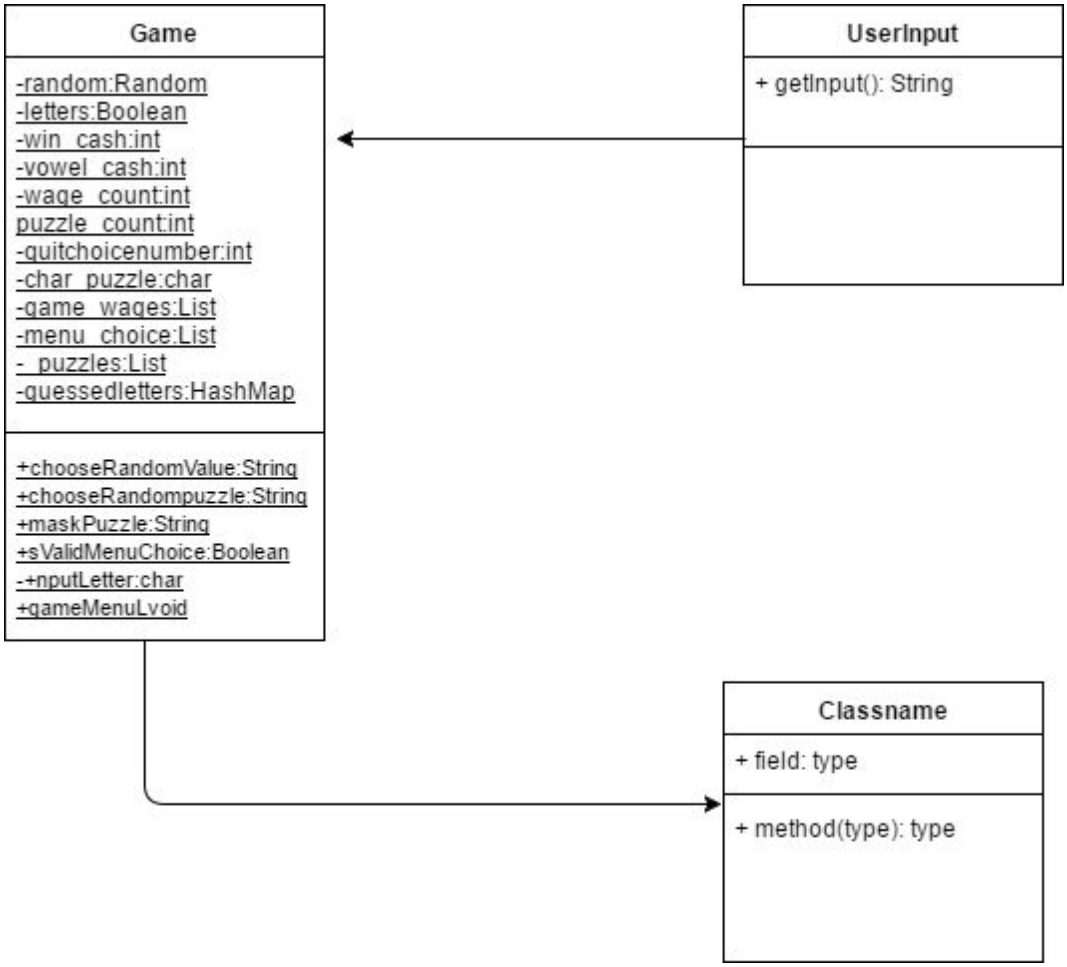
```
Enter choice: A
Invalid input
                    ----------------------
                    *        PUZZLE      *
                    ----------------------


-   - - - - - - -   - -   - - -   - - - - -

1. Spin the wheel
2. Buy a vowel
3. Solve the puzzle
4. Check your balance
5. Quit the game
Enter choice:
```

In the case that something besides one of the integers displayed is entered, a try and catch exception handling is implemented. The try block contains a block of program statements within which an exception might occur. A try block is always followed by a catch block, which handles the exception that occurs in associated try block.

```
System.out.print("Enter choice: ");
line = input.getInput();
try {
    // If the input was not an integer, then that error will be caught
    choice = Integer.parseInt(line);
} catch (NumberFormatException e) {
    // Error message, then go to the top of the loop
    System.out.println("Invalid input");
    win_cash = win_cash -50;
    //System.out.println("Remaining Cash is : $" +win_cash );
    continue;
}
```

When the input is not an integer, the try part of the function will detect that it is not an integer and the catch part of the function gives an output displaying that it was an invalid. This will continue to loop until a correct integer is inputted. Having the try and catch function implemented makes it easier to catch these errors.

UML Diagram:



Word count:1059