CIS 162 Project 4
Farkle (a dice game)

Due Date

- at the start of class on Monday, 3 December (be prepared for quick demo and zyBook test)

Before Starting the Project

- Read chapter 10 (ArrayList) and 13 (arrays)
- Read this entire project description before starting

Learning Objectives

After completing this project you should be able to:

- *use* an `ArrayList` to store Objects
- use an array to store primitive types
- *write* looping constructs (for-each and for)

Game Rules

The game supports multiple players using six dice.

- Players accumulate points each turn with the goal of 10,000
- roll all dice to start
- select one or more scoring dice and set them aside (see below)
- choose to pass the dice and bank your points or, choose to roll again and risk losing your subtotal
- player loses turn and subtotal points (Farkle) if no remaining dice can be scored
- all scoring combinations must be achieved in a single throw
- player can continue to roll if all six dice have been scored

Scoring Categories

Dice values do not need to appear in a particular sequence to qualify for any category.

- Straight using all six dice (1000 pts)
- Three unique pairs (1000 pts)
- Ones (100 pts for each)
- Fives (50 pts for each)
- 3 of a kind (or higher)

Dice Value	3 of a kind	4 of a kind	5 of a kind	6 of a kind
1	1000	2000	3000	4000
2	200	400	600	800
3	300	600	900	1200
4	400	800	1200	1600
5	500	1000	1500	2000
6	600	1200	1800	2400

Step 1: GVdie

Rather than writing your own Die class, we are providing a completed class for you. Create a new class in BlueJ called `GVdie` and delete all of the provided code. Copy and paste the provided code from (`GVdie.java`) into the newly created class. It should compile with no errors. Do not make any changes to this code.

A GVdie can be in one of three states: 1) selected, 2) scored or 3) available to roll. Dice start as available to roll. Dice are selected by the player clicking on them and they change color. The game automatically converts selected to dice and they change to gray. Dice can not be rolled or selected once they have been scored.

You only need to use the following methods but you are encouraged to read the source code to understand how it works.

```
GVdie d1 = new GVdie();          // instantiate a GVdie
d1.roll();                       // roll the die (1 – 6)
int val = d1.getValue();         // check current value
d1.setBlank();                   // set face to blank (value to 0)
d1.setSelected(true);            // mark die as selected (or not)
d1.setScored(true);              // mark die as scored (or not)
if(d1.isSelected())              // check if die is selected
if(d1.isScored())                // check if die is scored
```

Step 2: Player

Use your Player class from Project 3.

Step 3: Create a class called Farkle (60 pts)

Instance Variables

A class should contain several instance variables. Some instance members are not expected to change after given an initial value. It is good practice to define these as *final* and use ALL CAPS for the names (section 11.1). Provide appropriate names and data types for each of the private instance variables:

- an object of type `Player` that tracks the score and other information for the current player
- an `ArrayList` of `GVdie`
- a `private final` member for the number of dice
- `int tally[]` - an array of seven integers to keep track of dice values
- declare `final` members for each scoring category. For example:

```
private final int STRAIGHT = 1000;
private final int WINNING_SCORE = 10000;
```

Constructor

A *constructor* is a special method with the same name as the class and generally initializes the fields to appropriate starting values.

- `public Farkle()` - instantiate the `ArrayList` and fill it with six `GVdie`. Remember to instantiate each `GVdie` before you add it to the `ArrayList`. Instantiate the array of seven integers. Instantiate the player object. Invoke the private `resetGame()` method. Adapt the following starter code.

```
myDice = new ArrayList <GVdie> ();
tally = new int[7];

// create dice
for (int i=1; i<=NUM_DICE; i++){
    myDice.add(new GVdie());
}
```

Accessor Methods

An *accessor* method does not modify class fields. The names for these methods, which simply return the current value of a field, often begin with the prefix 'get'.

- `public Player getActivePlayer()` – return the current player object (one line).
- `publc boolean gameOver()` - return `true` if the game is over because the current player achieved at least 10,000 points. Otherwise, return `false`.
- `public ArrayList <GVdie> getDice ()` - return the `ArrayList` of `GVdie`. This method is only one line of code and is invoked by the GUI to display the dice.

Helper Methods

Designated as *private*, a helper method is designed to be used by other methods within the class. Good practice is to make methods private unless they need to be public.

- `private void tallySelectedDice()` – Update the array of integers to tally the number of 1s, 2s, 3s, 4s, 5s and 6s for the selected dice. Index zero of the array is not used. Remember to clear the array first. Adapt the following sample code.

```
// clear array
for (int i=1; i<tally.length; i++){
   tally[i] = 0;
}

// update tally for each selected GVdie
for (GVdie d : myDice){
   if(d.isSelected()){
      int val = d.getValue();
      tally[val]++;
   }
}
```

- `private void tallyUnscoredDice()` – Update the array of integers to tally the number of 1s, 2s, 3s, 4s, 5s and 6s for the dice that are not already scored. Remember to clear the array first.
- `private boolean hasStraight()` – assume dice have been tallied. Use the tally array to determine if the six dice contain a straight if each value (1-6) has one die. Return `true` or `false`. No scores are updated.
- `private boolean hasThreePairs()` – assume dice have been tallied. Use the tally array to determine if the six dice contain three unique pairs. Return `true` if they do. Otherwise, return `false`. No scores are updated.
- `private void nextTurn()` – this private helper method prepares for the next round by setting all dice to unscored, unselected and blank.

Mutator Methods

A mutator method performs tasks that may modify class fields. Refer to section 5.7.

- `public void resetGame()` - reset the player object by invoking its `newGame()`. Unselect all dice and set them to blank by invoking `nextTurn()`. Only two lines of code.
- `public void scoreSelectedDice( )` – first, invoke `tallySelectedDice()`. Next, check for each scoring category and update the player's subtotal when appropriate. Refer to the scoring category definitions at the start of this document. Afterwards, convert all selected dice to scored. This is a lengthy method! Give thought to the order that you check each category and test your solution thoroughly.
- `public void rollDice()` – score selected dice by invoking `scoreSelectedDice()`. If all dice have been scored, reset all dice to unselected and unscored. Roll each die not selected or scored. You will eventually add more logic to prevent cheating but this will at least make the game functional.
- `public void passDice()` – score selected dice by invoking `scoreSelectedDice()`. Have the player object update its score. Prepare for the next turn by invoking `nextTurn()`. Only three lines of code.

Step 4: Prevent Cheating (10 pts)

The game will be functional after completing step 3 but the player can cheat in a variety of ways. The following enhancements will help prevent most cheating.

- Define additional instance variables (boolean) to keep track if the player is allowed to roll, allowed to pass or if this is the initial roll.
- `public boolean okToRoll()` - return `true` if the player is allowed to roll. Otherwise, return `false`. One line of code.
- `public boolean okToPass()` - return `true` if the player is allowed to pass. Otherwise, return `false`. One line of code.
- `private boolean noDiceSelected()` – return `true` if no dice are selected. Otherwise, return `false`. Recall, dice are either selected, scored or neither.
- `public boolean playerFarkled()` – invoke `tallyUnscoredDice()`. Check for each scoring category to determine if any of the unselected dice could be used for scoring. The logic is similar to `scoreSelectedDice()` except no scores are updated. Instead, return `true` if no scoring category is possible among the unselected dice.
- Update methods as needed to ensure the state variables variables are correctly set to `true` or `false` throughout the game. For example, `nextTurn()` should set initial roll to `true`, OK to roll should be `true` and OK to pass should be `false`. What other methods might need updating?

  Player is always allowed to roll except when no scoring dice remain (Farkled).
  Player is always allowed to pass except at the start of a turn.
- Update `rollDice()` to only roll if it is the player's initial roll or at least one die is selected.

Step 5: Advanced Game Features

Support Multiple Players (5 pts)
You only need to add a few lines of code and one additional method to support multiple players.

- Add an instance field for an array of `Player` objects.
- Within the constructor, instantiate the players within the array and set the game's player object to array element 0. Modify the follow example to use your variable names.

```
players = new Player[3];
players[0] = new Player("Player 1");
players[1] = new Player("Player 2");
players[2] = new Player("Player 3");
thePlayer = players[0];
```

- `public void setActivePlayer(int id)` – set the game's current player object to array element `id` - 1. Correct player numbers should be 1 – 3. All future game actions and scores will relate to the current player.

```
thePlayer = players[id - 1];
```

- Update `resetGame()` to reset each player in the array rather than the current player. Adapt the following sample code;

```
for(Player p : players){
    p.newGame();
}
```

- No additional changes are needed anywhere in your code!

Support Testing (5 pts)
The following methods are provided for external testing only.

- `public void setAllDice (int [] values)` – pass an array of six integers to set the dice values. Repeatedly roll each die until the desired value is obtained. If a requested value is not between 1 and 6 then set it to one. This method can be less than ten lines of code with a nested loop. Test your solution thoroughly because it plays a critical role in our automated testing.
- `public void selectDie (int id)` – set the requested die to selected. Dice are numbered 1 – 6. For example, `selectDie(2)` will mark the 2nd die as selected. This method is one line of code. Recall, `ArrayList` indices start at zero.

Step 6: Best Score (10 pts)

Add a feature for the game to keep track of the player with the best score during the current session. The best score reflects a score above 10,000 with the fewest number of turns.

- Add another instance member of type `Player` to store the player with the best score.
- `public Player getBestPlayer ()` – return player with best score (one line).
- `public void setBestPlayer(Player p)` – set best player (one line). This method is used during automated testing.
- `private void checkBestScore()` – check if the winning player used a lower number of turns than the current best player. Best score is determined by the fewest number of turns, total score is ignored. Note: this method should only be invoked at the end of a game. So where would be a good choice within the `Farkle` class?

Saving Best Score (5 pts)

Add a feature for the best player information to be saved in an external data file. The best player information including name, score and number of turns is initially read from a file when the game starts (in the constructor). The updated player information is automatically saved to the data file. Read section 15.3 and refer to class notes. To support our automated testing, name the file `bestplayer.txt` using the following format.

```
Tiger Woods
10400
9
```

- `public void saveBestPlayer()` – save player information to data file
- `public void loadBestPlayer()` – read player information from data file

Coding Style (10 pts)

Good programming practice includes writing elegant source code for the human reader. Follow the GVSU Java Style Guide.

Step 7: Software Testing (10 pts)

Write JUnit tests for your `Farkle` class. Thoroughly testing your code requires *a lot* (dozens) of test cases. Don't try to write them all at the end --- you won't have enough time. Instead, test the methods as you write them.

Here are some guidelines to help you write tests:

- Make sure each line of code is run by at least one test
- Make sure each `if` statement is run by at least two tests: One for which the condition is true, and one for which it is false.
- Make sure you have a test for each way to score (straight, three pairs, three-of-a-kind, four-of-a-kind, etc.)
- Also write tests that verify that your code doesn't incorrectly award points for straights, and such.

```
@Test
public void testSetAllDice() {
    Farkle f = new Farkle();
    int[] expected = {2, 4, 5, 6, 1, 3};
    f.setAllDice(expected);

    int[] observed = new int[expected.length];
    for (int i = 0; i < expected.length; i++) {
        observed[i] = f.getDice().get(i).getValue();
    }
    Assert.assertArrayEquals(expected, observed);
}

@Test
public void correctlyScoresStraight() {
   Farkle f = new Farkle();
   f.setAllDice(new int[]{3, 2, 1, 6, 5, 4});
   for (int i = 1; i <= 6; i++) {
      f.selectDie(i);
   }
   f.scoreSelectedDice();
   Assert.assertEquals(1000, f.getActivePlayer().getSubtotal());
}
```

zyLab Testing

Upload `Farkle.java, Player.java` and `GVdie.java` to Ch 16 zyProject Farkle AFTER you are absolutely certainly that your code is correct and it passes ALL of your carefully written tests. You are limited to 10 submissions.

Step 8: GUI

Update `FarkleGUI` from Project 3. You should not have to change any of your code except the following:

- Replace the `FarkleStub` object with `Farkle`

Thoroughly test your GUI and be prepared to demo to your instructor

Grading Criteria

The project grade is based on the following:

- Program requirements (as specified above)
- Stapled cover page with your name and signed pledge. (-5 pts if missing)

Submission

A professional document **is stapled** with an attractive cover page. Do not expect the lab to have a working stapler!

- Cover page - Provide a cover page that includes your name, a title, and a screenshot of your GUI
- Signed Pledge – The cover page must include the following signed pledge: "I pledge that this work is entirely mine, and mine alone (except for any code provided by my instructor). " In addition, provide names of any people you helped or received help from. Under no circumstances do you exchange code electronically. You are responsible for understanding and adhering to the School of CIS Guidelines for Academic Honesty.
- Time Card – The cover page must also include a brief statement of how much time you spent on the project. For example, "I spent 7 hours on this project from January 22-27 reading the book, designing a solution, writing code, fixing errors and putting together the printed document."
- Sample Output – provide a screenshot of the GUI on your cover page
- Source code - a printout of your elegant source code.

```
Farkle.java
FarkleTest.java
```

- Demo – be prepared to demo your project on a lab computer or your laptop. Your instructor will ask you to perform a variety of tasks using BlueJ. You will also be asked to show your code passing the tests in zyLab.

Extra Credit: (+10 points)

Modify your GUI and Game class to handle an arbitrary number of players.