# CIS 162 Project 3

# Poker Dice

#### Due Date

At the beginning of lab on week 11 – See due date on Blackboard for your specific section.

Suggested guidelines for the amount of time required for each phase:

* Phase 1 – one – two days
* Phase 2 – one to one and a half week
* Phase 3 – Beginning of lab on week 11 - See due date on Blackboard for your specific section.

#### Before Starting the Project

* Read chapter 8 (about arrays)
* Read this entire project description before starting

#### Learning Objectives

After completing this project, you should be able to:

* *use* arrays to store primitive types and reference to Objects
* write methods to meet specific requirements
* *write* conditional statements with boolean expressions
* *write* looping constructs

#### Game Rules

The game supports a single player using five dice.

* each game consists of seven rounds
* each round consists of one, two or three rolls
* player rolls all five dice at the start of each round
* player may choose a subset of dice (1 or more) and roll a second time (optional). Dice are held by clicking on them and they become highlighted. Unselected dice are rolled again.
* player may choose a subset of dice (1 or more) and roll a third time (optional)
* player may stop at any time after the first, second, or third roll and select the most appropriate scoring category. Player may have to take a zero if no category is satisfied.
* each scoring category can be selected once, and only once
* the current score is displayed on the screen
* player may start a new game at any time by selecting File -> New Game
* player may quit at any time by selecting the File -> Quit

#### Scoring Categories

**NOTE: Dice values do not need to appear in a particular order to qualify for each category.**

* five-of-a-kind (50 pts) - the five dice have the same value
* four-of-a-kind (40 pts) - four of the dice have the same value
* Three-of-a-kind (25 pts) - three of the dice have the same value
* full house (35 pts) a pair and three-of-a-kind such as 2 2 4 4 4. A five-of-a-kind is also considered a full house
* a small straight using four of the dice (30 pts) – four of the dice have consecutive values, such as 1 2 3 4 or 2 3 4 5 or 3 4 5 6
* a large straight using five dice (45 pts) – the five dice have consecutive values such as 2 3 4 5 6 or 1 2 3 4 5
* chance (total of all dice) – the points should be equal to the sum of the values of all dice

#### Step 1: Create a New BlueJ Project

#### Step 2: Use Existing GVdie

Rather than writing your own Die class, we are providing a completed class for you. Save the provided GVdie class in the folder that BlueJ created for your project. It should compile with no errors. **Do not make any changes to this code.** You only need to use the following methods but you are encouraged to read through the source code to see how it works.

GVdie d1 = new GVdie(); // declare and instantiate a GVdie

d1.roll(); // roll the die

int val = d1.getValue(); // check current value

d1.setBlank(); // set face to blank

d1.setHeld(true); // mark die as selected

d1.setHeld( ??? ); // which parameter to unselect the die?

if(d1.isHeld()) // check if die is selected

# Phase 1 (20 pts)

#### Step 3: Create a class called PokerDice

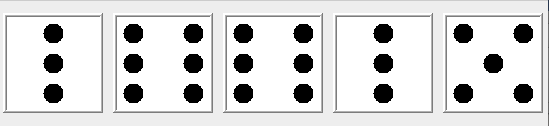
Exact spelling is required for the name of the PokerDice class and the headers of all the methods within the PokerDice class. Do not make any changes to the following requirements. If you do so, the automated tests that we provide you (see below) will likely fail.

**Instance Variables**

* **an array of five GVdie objects**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| GVdie | GVdie object | GVdie object | GVdie object | GVdie object | GVdie object |
| index | 0 | 1 | 2 | 3 | 4 |

Example



* **integers to keep track of the score, number of rolls, and number of rounds**
* **an array of seven integers used by the helper methods. This array is used to tally the number of 1s, 2s, 3s, 4s, 5s and 6s for the current dice.**

**Important note:** Index zero of the array is not used

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| int  counts | Not used | count of 1’s | count of 2’s | count of 3’s | count of 4’s | count of 5’s | count of 6’s |
| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

* **declare private final static members for each of the scoring categories. We are providing the declaration of two constants for the scoring categories, you need to include the declaration of the constants for the rest of the scoring categories.**

**private final static int FIVE\_OF\_A\_KIND = 50;**

**private final static int FOUR\_OF\_A\_KIND = 40;**

#### Constructor

* **public PokerDice ( ) –**
* **instantiate and populate the array of GVdie objects**
  + - **instantiate the array of five GVdie objects**
    - **use a loop to instantiate the five GVdie objects and assign them to the elements of the array**
* **instantiate the array of seven integers.**
* **invoke the resetGame() method to initialize the instance variables. You will be writing the resetGame method later in the program.**

#### Accessor Methods

* **public int getScore ( ) - return the current score.**
* **public int getNumRolls ( ) – return number rolls**
* **public int getNumRounds ( ) – return number rounds**
* **public boolean okToRoll ( ) - return true if it is legal to roll the dice. Otherwise, return false. Only three rolls per round are allowed.**
* **publc boolean gameOver ( ) - return true if the game is over because all seven rounds have been completed. Otherwise, return false.**
* **public GVdie [] getDice ( ) - return the array of GVdie. This method is only one line of code and is invoked by the GUI so that it can display the game's dice.**

#### Mutator Method

* **public void resetGame ( ) –** 
  + **set the score, number of rolls, and number of rounds to zero**
  + **use a loop to set all dice to blank** and make sure **ALL** dice are unselected (not held)

#### Software Testing – PokerDiceTest class (20 pts)

For this project, write a **PokerDiceTest** class that contains a main method.

The following code is a brief **and incomplete** example. You can use this code as a starting point. You should add logic to invoke the methods to simulate a game: roll the dice and check the different game categories.

Notes:

* To be able to test the functionality of each phase, you will add instructions to the main method in each phase.
* It takes careful consideration to anticipate and test every possibility.

public static void main () {

final int THREE\_OF\_A\_KIND = 25;

final int FOUR\_OF\_A\_KIND = 40;

final int FIVE\_OF\_A\_KIND = 50;

final int FULL\_HOUSE = 35;

final int SMALL\_STRAIGHT = 30;

final int LARGE\_STRAIGHT = 45;

int oldScore;

System.out.println ("Testing begings");

PokerDice game = new PokerDice();

//\*\*\*\*\*\*\*\*\*\* phase 1 testing \*\*\*\*\*\*\*\*\*\*\*\*

// testing the constructor

assert game.getDice().length == 5 : "dice should be an array of five";

assert !game.gameOver() : "game should not be over at the beginning";

//\*\*\*\*\*\*\*\*\*\* phase 3 testing \*\*\*\*\*\*\*\*\*\*\*\*

// testing fiveOfAKind - OK

game.setDice (new int [] {2,2,2,2,2});

game.checkFiveOfAKind();

assert game.getScore() == FIVE\_OF\_A\_KIND : "Score should increment by " + FIVE\_OF\_A\_KIND ;

// testing fiveOfAKind - NOT OK

oldScore = game.getScore();

game.setDice (new int [] {2,6,2,2,2});

game.checkFiveOfAKind();

assert game.getScore() == oldScore : "Not fiveOfAKind " ;

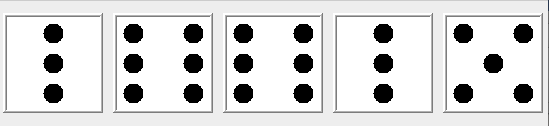
System.out.println ("Testing ends");

}

# Phase 2 (30 pts)

* **public String diceToString ( ) – return (don’t print) a formatted String of the array of GVdie objects. You can use this method in your test class to be able to see the values of your dice.**
  + **use a loop to navigate the array of dice**
  + **inside of the loop get the value of each dice and add it to the String that is being returned. Use a comma to separate the values of the dice.**

**Example:** If the values of the dice are:



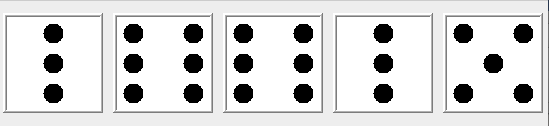
The method will return the String: " [3,6,6,3,5] "

#### Helper Methods

Designated as *private*, a helper method is designed to be used by other methods within the class.

* **private void tallyDice( ) – this private helper method supports the other methods. Update the array of integers to tally the number of 1s, 2s, 3s, 4s, 5s and 6s for the current dice. Index zero of the array is not used.** 
  + **use a loop to clear the array used to tally the numbers**
  + **use a loop to navigate the array that contains the 5 dice. Inside of this loop, get the value of each die and use it as the index in the array of tallies to increment the count for that specific number**

Example: for the following dice array



The values of the tally array after invoking the tallyDice method should be:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| counts | Not used | 0 | 0 | 2 | 0 | 1 | 2 |
| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

* **private boolean hasStraight(int length) – this private helper method supports other methods. First, invoke the tallyDice() method. Return true if the set of dice contains a sequence of length numbers. Return false otherwise. This method will be invoked to check if there is a small or large straight. Note: Your logic will be using the tally array to figure out if the set of dice contains a sequence of length numbers.**
* **private boolean hasMultiples(int count) – this private helper method supports other methods. First, invoke the tallyDice() method. Return true if there are count, or more, identical values. Otherwise, return false. This method will be invoked to check for three, four and five of a kind. Note: write your logic using the tally array.**
* **private boolean hasStrictPair() – this private helper method supports other methods. First, invoke the tallyDice() method. Then, determine if the set of dice contains a pair of any numbers (but only a pair). Return true or false. Note: write your logic using the tally array.**
* **private void nextRound () – this private helper method prepares for the next round by incrementing the round counter, setting rolls count to zero, setting all dice to blank and unselected. This method is invoked by each of the scoring methods.**

#### For Testing Purposes

The following method is provided for testing purposes only. It allows you to test the private methods above by forcing values on each of the five dice. For example, you can force a roll of “1 2 3 4 5” and then test to see if your hasStraight(5) method returns true.

* **public void setDice (int [] values) – pass an array of five 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 and includes a nested loop.**

# Update PokerDiceTest to test Phase 2

Update your PokerDiceTest class to thoroughly test the new methods you just added.

**Note:** Most of the methods in phase 2 are private methods. To be able to test these methods, change the visibility to public. Once the methods work correctly, change the visibility back to private.

# Phase 3 (20 pts)

#### Mutator Methods

* **public void rollDice ( ) – use a loop to roll each of the dice that are not currently held.**
* **public void checkThreeOfAKind ( ) - update the score if the dice currently include a three of a kind. Invoke the hasMultiples(3) method to determine if the category is achieved. Afterwards, invoke the nextRound() method.**
* **public void checkFullHouse ( ) - update the score if the dice currently include a full house. Invoke the hasMultiples(3) and hasStrictPair() methods to determine if the category is achieved. Note, a five of a kind qualifies as a full house. Afterwards, invoke the nextRound() method to prepare for the next round.**
* **public void checkSmallStraight ( ) - update the score if the dice currently include a small straight. Use the hasStraight(4) method to determine if the category is achieved. Afterwards, invoke nextRound() to prepare for the next round.**
* **public void checkLargeStraight ( ) - update the score if the dice currently include a large straight. Use the hasStraight(5) method to determine if the category is achieved. Afterwards, invoke nextRound() to prepare for the next round.**
* **public void checkFiveOfAKind ( ) - update the score if the dice currently include a five of a kind. Invoke the hasMultiples(5) method to determine if the category is achieved. Afterwards, invoke nextRound() to prepare for the next round.**
* **public void checkFourOfAKind ( ) - update the score if the dice currently include a four of a kind. Invoke the hasMultiples(4) method to determine if the category is achieved. Afterwards, invoke nextRound() to prepare for the next round.**
* **public void checkChance ( ) - update the score with the sum of all dice. Afterwards, invoke nextRound() to prepare for the next round.**

# Update PokerDiceTest to test Phase 3

Update your PokerDiceTest class to thoroughly test the new methods you just added.

#### Coding Style (10 pts)

Good programming practice includes writing elegant source code for the human reader. Follow the GVSU [Java Style Guide](http://www.cis.gvsu.edu/java-coding-style-guide/).

#### Step 4: Software Testing

Our unit test file is not a substitute for your own tests. We strongly suggest you thoroughly test your PokerDice with your PokerDiceTest class first. Then when you think you have everything right, then try our unit test class.

#### JUnit Testing

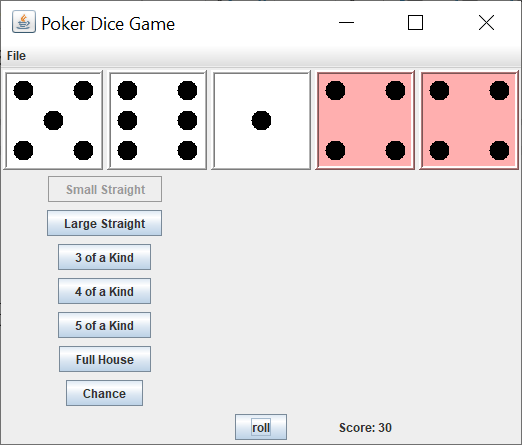
As an alternative, JUnit is a Java library that helps to automate software testing. A JUnit test file MyPokerDiceJunit.java has been provided on BlackBoard. Use the instructions from project 2 for additional information about JUnit Testing.

#### Step 5: Use the Graphical User Interface to play the game.

Now that you have the basic game working within its own class it is time to use a more interesting graphical user interface for the player to use.

Rather than writing your own GUI class, we are providing a completed class for you. Save the provided PokerDiceGUI class in the folder that BlueJ created for your project. It should compile with no errors.

**Sample GUI**  
There are no messages printed to the terminal window. All results appear within the GUI.



**Figure 1. Sample GUI. Pink dice are 'held' and not rolled.**

Grading Criteria

There is a 50% penalty on programming projects if your solution does not compile.

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

#### Late Policy

Projects are due at the START of the class period. However, you are encouraged to complete a project even if you must turn it in late.

* The first 24 hours (-20 pts)
* Each subsequent weekday is an additional -10 pts
* Weekends and university holidays are free days.

#### Turn In

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

* Cover page - Your project must have a cover page that includes your name, a title, an interesting graphic or photograph related to the project topic and the following signed pledge: "I pledge that this work is entirely mine, and mine alone (except for any code provided by my instructor). " You are responsible for understanding and adhering to the [School of CIS Guidelines for Academic Honesty](http://www.cis.gvsu.edu/Academics/Honesty/).
* Sample Output – a printout of the Terminal window showing a variety of the printed messages when running the main method of your PokerDiceTest class. You can cut and paste into the Word document that contains your cover page.
* Source code – DO NOT PRINT – upload to BlackBoard the following files: PokerDice.java and PokerDiceTest.java
* Demo - be prepared to demo your game to me by playing the game via the GUI. If you don’t demo your project, I will assume it doesn’t compile.