CIS 162 Project 2
Chuck A Luck: A Dice Game

Due Date
Week 11 – See due date on Blackboard for your specific section.
Suggested phases:

- Phase 1 – one day
- Phase 2 – one week
- Phase 3 – one week

Before Starting the Project

- Read chapter 8
- Read this entire project description before starting

Learning Objectives
After completing this project, you should be able to:

- *write* methods to meet specific requirements
- *write* code using arrays
- *write* conditional statements with boolean expressions
- *describe* the difference between public and private methods
- *read and understand* source code for a Graphical User Interface (GUI)

Game Rules
Chuck-a-luck is played with three standard 6 sided dice. Bets are placed on possible combinations that can appear on the dice. A player's turn includes making one or more bets and then rolling the dice. Each bet costs one credit. Different payouts are provided for winning each bet. A game begins with 10 credits.

<u>The various winning outcomes are described below:</u>

Large –	total value of the three dice is over ten AND there is no three-of-a-kind (player wins 2 credits)
Small –	total value of the three dice is under eleven AND there is no three-of-a-kind (player wins 2 credits)
Field –	total value of the three dice is under eight OR over twelve (player wins 2 credits)
1s –	the player wins 2 credits for throwing a single 1, 3 credits for a pair of 1s, and 11 credits for three-of-a-kind (three 1's)
2s –	same as above except the player's goal is to roll 2s.
3s –	same as above except the player's goal is to roll 3s.
4s –	same as above except the player's goal is to roll 4s.
5s –	same as above except the player's goal is to roll 5s.
6s –	same as above except the player's goal is to roll 6s.

Step 1: Create a New BlueJ Project

Step 2: Class Definition: GVdie

Rather than writing your own Die class, we are providing a completed class for you. Download `GVdie.java` to the folder of 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.

```
// declare and instantiate an object of the GVdie class
GVdie d1 = new GVdie();

// roll the die
d1.roll();

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

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

Phase 1 (10 pts)

Step 3: Class Definition: Chuck

Create a new class called `Chuck`. Implement each of the following methods. All must be complete before the game can be played.

Important Note:

We are providing a JUnit class to test your project. Exact spelling is required in the `Chuck` class for the class name and all the method headers. Do not change the method headers in any way.

All feedback is provided to the player through a single String instance variable that contains the current message. Nothing is printed to the screen. Your source code should have no `System.out.print()` statements (except for possible debug statements).

Instance Variables

A class should contain several instance variables. Provide appropriate names and data types for each of the private instance variables:

- an array of three `GVdie` objects.
- credit balance (integer)

- current message (String)
- an array of nine boolean values (stores the possible nine bets)

Visual representation of the two arrays

Array of three GVdie objects

	GVdie object	GVdie object	GVdie object
index	0	1	2

Array of nine boolean bets

bet	1's	2's	3's	4's	5's	6's	Field	Small	Large
boolean value	false	false	false	false	false	false	false	false	false
index	0	1	2	3	4	5	6	7	8

Constructors

- `public Chuck ( )` –
    - instantiate and populate the array of GVdie objects
        - instantiate the array of three GVdie objects
        - use a loop to instantiate the three GVdie objects and assign them to the elements of the array
    - instantiate the array of nine boolean variables
    - initialize the credit to 10 – (a game begins with 10 credits)
    - use a loop to set the three dice faces to blank.

Accessor Methods

- `public String getMessage ( )` - return the internal message.
- `public int getCredits ( )` - return the current credit balance.
- `public GVdie [] getDice ()` - return the array of the GVdie objects.

Software Testing – ChuckTest class (10 points)

For this project, write a **`ChuckTest`** class that contains a main method to test the Chuck class. For phase 1 test the constructor and the accessor methods.

Phase 2 (40 pts)

Private Helper Methods

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

- `private int  getSumDiceValues ()` – return the sum of the values of the three dice. Use a loop to navigate/iterate the array of GVdie objects
- `private boolean isDoubles (int num)` – return `true` if two of the dice values match `num`, return `false` otherwise. Use a loop to navigate the array of GVdie objects
- `private boolean isTriplets ( )` – return `true` if all three values of the dice are identical, return `false` otherwise.
- `private void checkLarge( )` – invoke the `getSumDiceValues()` method to get the sum of the values of the three dice, check if the sum is greater than 10 and there is no three-of-a-kind (triplet). If so, increase the credits by 2 and update the message to indicate that the player won. Note, invoke the `isTriplets()` method when appropriate.
- `private void checkSmall( )` – invoke the `getSumDiceValues()` method to get the sum of the values of the three dice, check if the sum is less than 11 and there is no three-of-a-kind (triplet). If so, increase the credits by 2 and update the message to indicate that the player won. Note, invoke the private `isTriplets()` method when appropriate.
- `private void checkField( )` – invoke the `getSumDiceValues()` method to get the sum of the values of the three dice, check if the dice total is less than 8 or greater than 12. If so, increase the credits by 2 and update the message to indicate that the player won.
- `private void checkNumber(int num)` – Check how many of the dice values match `num`. If all three dice match, increase the credits by 11. If only two dice match, increase the credits by three. If only one die matches, increase credits by 2. Invoke the `isTriplets` and `isDoubles` methods to keep this method as short as possible. For all wins, update the message to indicate that the player won.
- `private void checkAllBets ( )` – Initially set the message string to indicate that the player lost. The message will get reset in the helper methods if the player wins. Use a loop to navigate the array of bets. For each bet that was placed, decrease the number of credits by 1. This method invokes the appropriate helper methods to check if the bet was won. <u>For example</u>, if there is a bet placed associated with index 6, you need to invoke the `checkField` method, if there is a bet associated with the index number between 0 and 5 (inclusive) you need to invoke the `checkNumber` method. <u>Note:</u> pay attention to the input parameter that you need to pass when invoking the `checkNumber` method, remember that the array of bets starts at index 0.

Phase 3 (40 pts)

Mutator Methods

- `public void addCredits (int amount)` – add amount to the credits. Ignore the request if `amount` is negative.
- `public void placeBet (int bet)` - Set the appropriate element of the array of bets to `true` based on the bet parameter. The valid values of the bet parameters are between 1 and 9 (inclusive). Remember that the index of the array starts at 0.

  <u>Example:</u>

  Visual representation of the array of bets when there are bets placed for 1's, 2's and large

bet	1's	2's	3's	4's	5's	6's	Field	Small	Large
boolean value	`true`	`true`	`false`	`false`	`false`	`false`	`false`	`false`	`true`
index	0	1	2	3	4	5	6	7	8

- `public void cancelBet (int bet)` - Set the appropriate element of the array of bets to `false` based on the bet parameter. The valid values of choice are between 1 and 9 (inclusive). Remember that the index of the array starts at 0.
- `public void clearAllBets( )` – Use a loop to set all the elements of the bets array to `false`.
- `public void roll ( )` -
  - Use a loop to navigate the array of GVdie objects and roll the three dice.
  - Invoke the `checkAllBets()` method to check each of the bets to determine which bets, if any, the player won.
  - Invoke the `clearAllBets()` method to clear all bets to prepare for the next round.
- `public void reset ()` – reset for a fresh game.
  - Use a loop to set all dice to blank.
  - Set the message and credits to the appropriate values
  - Invoke the `clearAllBets( )` method to clear all bets

Preventing Player Errors

The game is now functional but it is possible the player will make various errors. Make the following improvements to prevent common errors.

- `public boolean enoughCredits ( )` – return `true` if the player has enough credits to cover all bets, return `false` otherwise. Use a loop to navigate the array of bets and count the number of active bets and compare it with available credits.
- Modify the `roll()` method to only roll the dice if there are enough credits to cover all current bets. Invoke the `enoughCredits ()` when appropriate. Otherwise, update

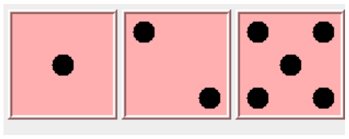
the message to inform the player that there are not enough credits to play and set the dice to blank.

Step 4: Software Testing

Games and other applications that generate random results are difficult to test. The following methods are provided within the Chuck class to help during testing. It would normally be removed after testing is complete but you will keep it in your solution. These two methods will be invoked in the JUnit class provided to do the automatic testing.

- `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.

  <u>Example:</u> If the values of the dice are:

  The method will return the String: [1,2,5]

- `public void testRoll(int [] values)` – this method will roll the dice until the desired values entered as the array of 3 integer values have been rolled. This makes the testing a lot easier because you can control what numbers are rolled.
  - If there are enough credits to play, this method has to do the following:
    - If a requested value is not between 1 and 6 then set it to one.
    - Repeatedly roll each die in the array of dice until the desired value is obtained.

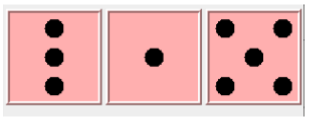
      <u>Notes:</u>

      This logic includes a nested loop.

      <u>Do not make any changes to the GVdie class</u>. The GVdie class does not have a setValue method to avoid cheating.

      <u>Example:</u>

      If the array entered as input parameter is {3,1,5}, this method will roll the die in position zero until its value is 3, etc.

- Now that the dice have the wanted values
  - invoke the `checkAllBets()` method to check each of the bets to determine which bets, if any, the player won.
  - invoke the `clearAllBets()` method to clear all bets to prepare for the next round.

Update ChuckTest – update the main method

Update the main method in the ChuckTest. Instantiate a game and invoke the methods with a variety of parameter values to test each method. It takes careful consideration to anticipate and test every possibility.

This is an incomplete example. **Your solution will be longer.**

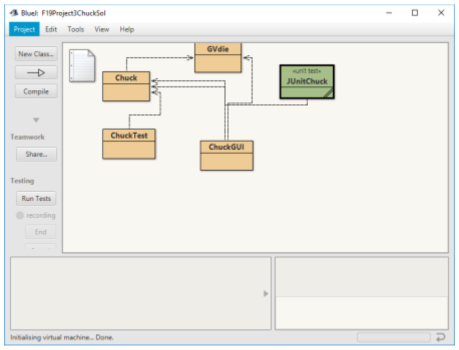
Main Method for Testing

```
    public static void main(String [] args){
        int before = 0;

        final int ONES = 1;
        final int TWOS = 2;
        final int THREES = 3;
        final int FOURS = 4;
        final int FIVES = 5;
        final int SIXES = 6;
        final int FIELD = 7;
        final int SMALL = 8;
        final int LARGE = 9;

        Chuck game = new Chuck();

        System.out.println("Testing begins...");
        assert game.getCredits() == 10 : "credits should start at 10";

        // wins bet on Large
        before = game.getCredits();
        game.placeBet(LARGE);
        game.testRoll(new int [] {6,3,3});
        assert game.getCredits() == before + 1 : "should have won betting on Large";

        // loses bet on Large
        before = game.getCredits();
        game.placeBet(LARGE);
        game.testRoll(new int [] {2,3,3});
        assert game.getCredits() == before - 1 : "should have lost betting on Large";

        System.out.println("Testing completed.");
    }
```

JUnit Testing

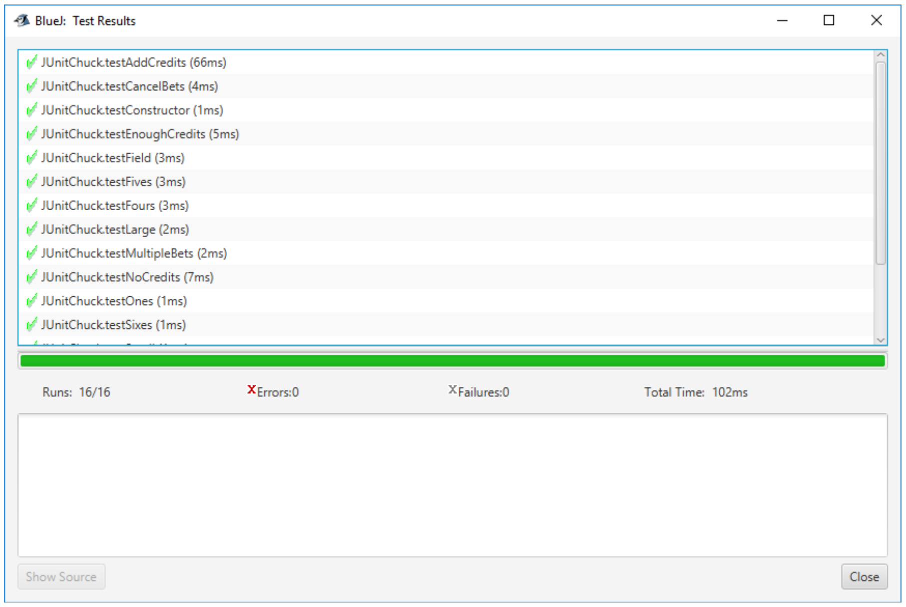
JUnit is a Java library that helps to automate software testing. A JUnit test file has been provided on Blackboard. Follow these instructions to use the test file.

1. Name your class `Chuck`. **Exact spelling is required** in the `Chuck` class for the class name and all the method headers.
2. Complete ALL requirements described above or some of the unit tests will fail.
3. Download `JUnitChuck.java` from Blackboard to the same folder as your `Chuck.java`
   Restart BlueJ (if `JUnitChuck` does not show up on your BlueJ window). BlueJ should recognize `JUnitChuck` as a "<<unit test>>" file.
4. On the BlueJ window, click the "Run Tests" button.



<u>Example: JUnit test passes all the test cases</u>

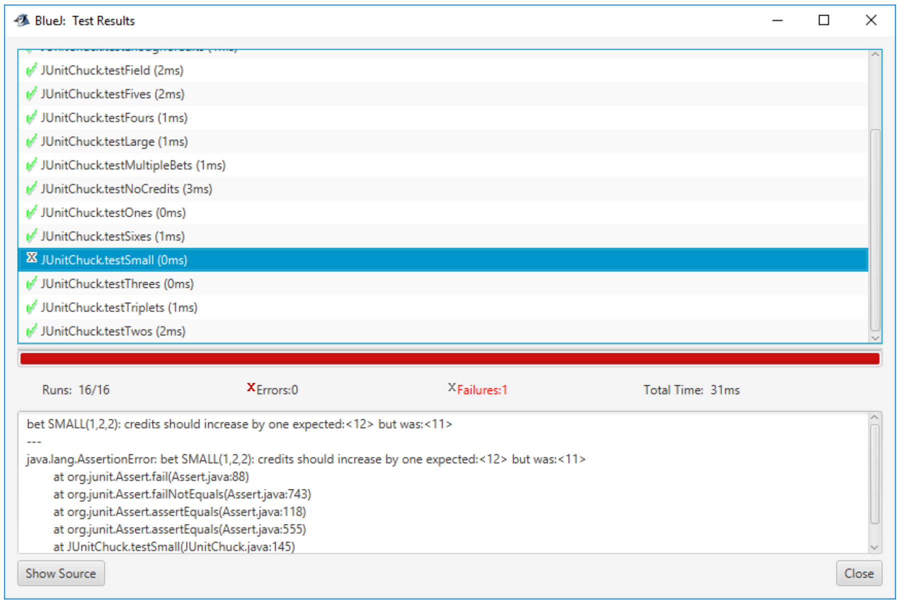
If your class passes all the test cases, the test results window will show a green horizontal bar and zero failures.


BlueJ: Test Results

```
✓ JUnitChuck.testAddCredits (66ms)
✓ JUnitChuck.testCancelBets (4ms)
✓ JUnitChuck.testConstructor (1ms)
✓ JUnitChuck.testEnoughCredits (5ms)
✓ JUnitChuck.testField (3ms)
✓ JUnitChuck.testFives (3ms)
✓ JUnitChuck.testFours (3ms)
✓ JUnitChuck.testLarge (2ms)
✓ JUnitChuck.testMultipleBets (2ms)
✓ JUnitChuck.testNoCredits (7ms)
✓ JUnitChuck.testOnes (1ms)
✓ JUnitChuck.testSixes (1ms)

Runs: 16/16        Errors:0        Failures:0        Total Time: 102ms
```

<u>Example: JUnit test does not pass all the test cases:</u>

If your class does NOT pass all the test cases, you will see a red horizontal bar and the number of failures. Click on a test result to show a longer description of the failure.



```
✓ JUnitChuck.testField (2ms)
✓ JUnitChuck.testFives (2ms)
✓ JUnitChuck.testFours (1ms)
✓ JUnitChuck.testLarge (1ms)
✓ JUnitChuck.testMultipleBets (1ms)
✓ JUnitChuck.testNoCredits (3ms)
✓ JUnitChuck.testOnes (0ms)
✓ JUnitChuck.testSixes (1ms)
✗ JUnitChuck.testSmall (0ms)
✓ JUnitChuck.testThrees (0ms)
✓ JUnitChuck.testTriplets (1ms)
✓ JUnitChuck.testTwos (2ms)

Runs: 16/16        ✗Errors:0        ✗Failures:1        Total Time: 31ms

bet SMALL(1,2,2): credits should increase by one expected:<12> but was:<11>
---
java.lang.AssertionError: bet SMALL(1,2,2): credits should increase by one expected:<12> but was:<11>
	at org.junit.Assert.fail(Assert.java:88)
	at org.junit.Assert.failNotEquals(Assert.java:743)
	at org.junit.Assert.assertEquals(Assert.java:118)
	at org.junit.Assert.assertEquals(Assert.java:555)
	at JUnitChuck.testSmall(JUnitChuck.java:145)
```

Coding Style (10 pts)

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

Step 5: GUI class

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

NOTE: This step does not require you to do anything except downloading the GUI and playing the game!

Rather than writing your own GUI class, we are providing a complete class. Download the `ChuckGUI.java` from Blackboard to the same folder as your `Chuck.java`. It should compile with no errors. If it doesn't compile then it probably means you used different method names in your `Chuck` class than what was specified.

We modeled our design on examples from the book in Section 10.3. Read the code and comments carefully to understand how it works. To play your game invoke the GUI's `main` method.

Have fun playing your game and testing your program!

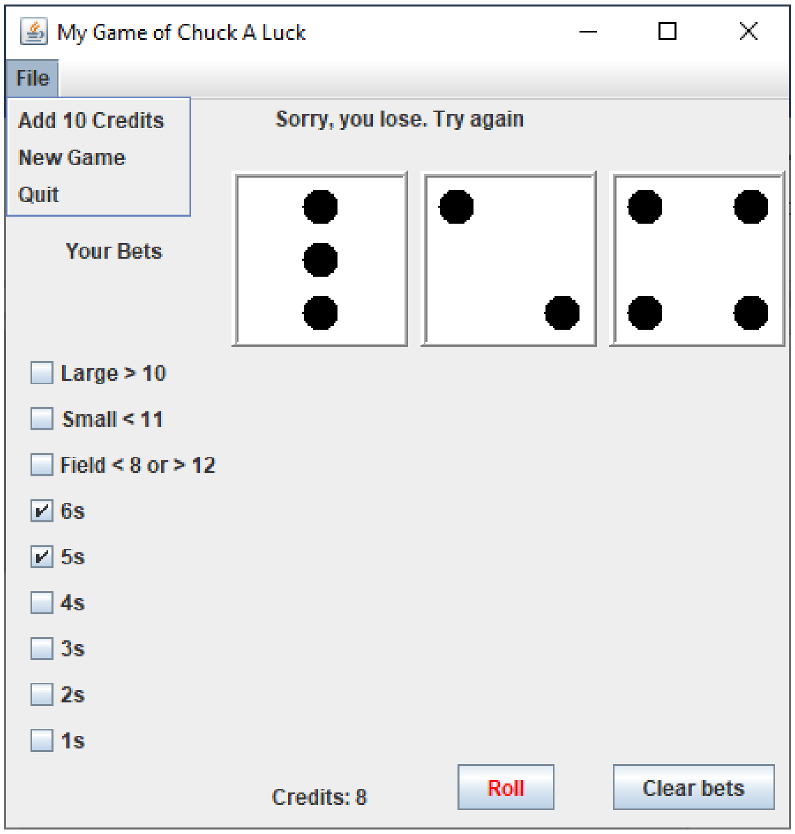


Figure 1. GUI screen shot.

Grading Criteria

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

- Cover page with your name and signed pledge. (-5 pts if missing)
- Project requirements as specified above. (90 pts)

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

1. A Word document that includes:
   - Cover page - Provide a cover page that includes your name, a title, and an appropriate picture or clip art for the project
   - 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 – a printout of the Terminal window after running the main method that shows a variety of the printed messages. You can copy and paste into the Word document that contains your cover page.
   - A screenshot after you run the JUnit test class JUnitChuck.java showing you passed all the test cases.
   - A screenshot of your GUI working

2. Source code – DO NOT PRINT - upload to Blackboard the source code of the two classes. `Chuck.java` and `ChuckTest.java`