**CIS 162 Project 3**

**A Game of Pig**

**Due Date**

* Thursday, June 18, 2015 at the beginning of class.

**Before Starting the Project**

* Read chapter 4 (branching), chapter 5 (boolean variables) and chapter 6 (while loops)
* Read this entire project description before starting

**Learning Objectives**

After completing this project you should be able to:

* *use* a Graphical User Interface (GUI)
* *write* methods to meet specific requirements
* *write* conditional statements with boolean expressions
* *write* a looping construct
* *write* programs that include the use of several classes.

**Game Rules**

The game requires two six-sided dice.

Players take turns until someone wins by earning at least 100 points

Players begin a turn by rolling both dice and tallying the points. The current player continues rolling and accumulating points until rolling a '1' or choosing to 'hold'. If the player rolls a 1 then all points from this round are lost. If the player rolls a pair of 1s then all player points are lost.

Play rotates to the next player

Your game will be between a person and the computer. The player always goes first.

**Step 1: Create a New BlueJ Project – Project3**

**Step 2: Class Definition: GVdie**

Rather than writing your own Die class, we are providing a completed class for you. Download the GVdie class and use the <edit> <Add Class from File> from BlueJ to include the GVdie class in the Project3 project created in step 1. It should compile with no errors.

You will not need to make any changes to the code of the GVdie class but will need to understand how to use each of the following methods:

* public void roll ( ) – randomly change the value of the die (1 - 6).
* public int getValue ( ) – returns the current value of the die.

**Step 3: Create a class called Pig (60 pts)**

* Provide appropriate names and data types for each of the instance variables. Maintain two GVdie objects, player’s score, computer’s score, current round score and a final int for the winning score of 100.
* public Pig ( ) - a constructor that initializes all of the instance variables to appropriate values, *instantiates* two dice of the GVdie class and prints a welcome message.
* public int getRoundScore ( ) – return the current round score.
* public int getPlayerScore ( ) – return the player’s score.
* public int getComputerScore ( ) – return the computer’s score.
* private void rollDice ( ) - roll both dice by invoking the roll method of the GVdie class for each die. If either value is '1' then set the round score to 0. Otherwise, increment the round score by the new total. Print the round score. This method is used by both the player and the computer.
* public void playerRolls ( ) - performs the first half of the player turn: 1) invoke the rollDice( ) method, 2) print the player's score if the round is over, or 3) print an appropriate message if the player wins. Refer to the sample output below. Set the player score to zero if double 1s are rolled.
* public void playerHolds ( ) - performs the second half of the player's turn: 1) update the player's score, 2) reset the round score to zero and 3) print the player's score. Refer to the sample output below.
* public void computerTurn ( ) - performs the computer's entire turn: continue rolling the dice and updating the round score until: 1) a '1' is rolled or 2) the round score surpasses 19 or 3) the computer wins. Use a while loop to repeatedly roll the dice. Set the computer score to zero if double 1s are rolled. Update the computer's score and reset the round score to zero. Print the computer's score. Refer to the sample output below.
* public void restart ( ) – reset all instance variables to start a new game. **Do not** instantiate new dice objects.
* public GVdie getDie (int num) - return the requested die. Legal values for the parameter are 1 or 2.
* public boolean playerWon ( ) - return true if the player score is currently high enough to win. Otherwise, return false.
* public boolean computerWon ( ) - return true if the computer score is currently high enough to win. Otherwise, return false.

**Step 4: Software Testing (15 pts)**

Software developers must plan from the beginning that their solution is correct. BlueJ allows you to instantiate objects and invoke individual methods. You can carefully check each method and compare actual results with expected results. However, this gets tedious. Another approach is to write methods that automatically play a game and therefore call all of the other methods. You can carefully review the results of a game to confirm that it works correctly.

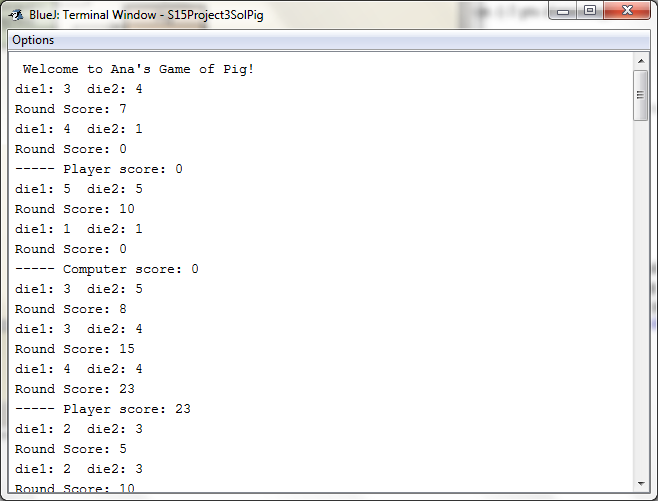
* public void playerTurn ( ) - simulates the player's entire turn: continue rolling the dice and updating the round score until a '1' is rolled or the round score surpasses 19 or if the player wins. . Set the player score to zero if double 1s are rolled. Update the player's score and reset the round score to zero. Print the player's score.
* public void autoGame ( ) - automates an entire game by alternating between a player's turn and computer's turn until one of them wins. Keep track and report the total number of turns taken. Report who wins. Reset all values before starting the new game.

**JUnit Testing**

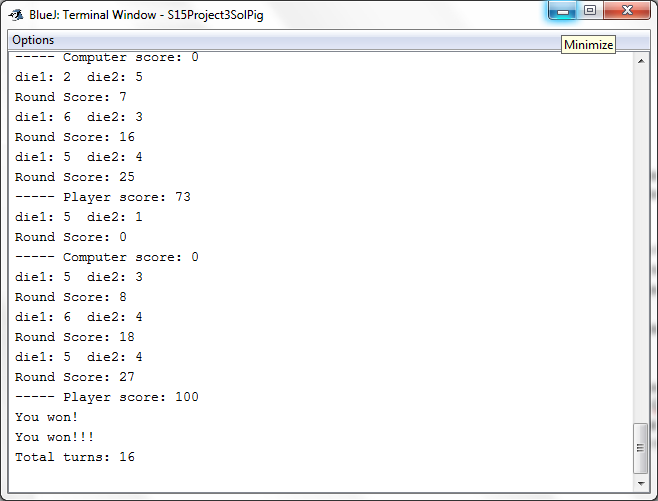
As an alternative, JUnit is a Java library that helps to automate software testing. A JUnit test file PigTest.java has been provided on BlackBoard. See explanation of JUnit below.

1. Name your class Pig. **Exact spelling is required**. Using a different name will fail the tests.
2. Complete all methods described in Step 3 and Step 4.
3. Download PigTest.java to the same folder as your Pig.java
4. Restart Bluej (if PigTest does not show up on your BlueJ window). BlueJ should recognize PigTest as a “<<unit test>>” file.

**Sample Results**  
The following sample shows partial results of a game. Your messages can be more creative as long as they convey the necessary information. Note: totals turns is only displayed during automated games.



*later in the game…*



**Step 5: Adapt for a GUI (15 pts)**

Now that you have the basic game working within it’s own class it is time to create a more interesting graphical user interface for the player to use. Add the following methods to the Pig class.

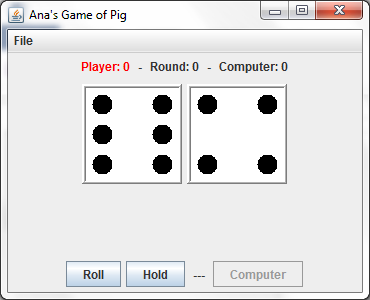
* Add a boolean instance variable to indicate if it is currently the player’s turn. You will need to make several changes throughout the Pig class to set this variable to true when it is the player’s turn and false when it is the computer’s turn. All games start with the player.
* public boolean isPlayerTurn ( ) - this one line method returns true if it is the player's turn or false if it is the computer's turn.

Rather than writing your own GUI class, we are providing the complete class. Download the GVdie class and use the <edit> <Add Class from File> from BlueJ to include the GVdie class in the Project3 project created in step 1. It should compile with no errors. If it doesn’t compile then it probably means you used different names for your methods than what was specified.

Make the following change to the GUI class

* Change the JFrame title to include your name. See comment in the code.

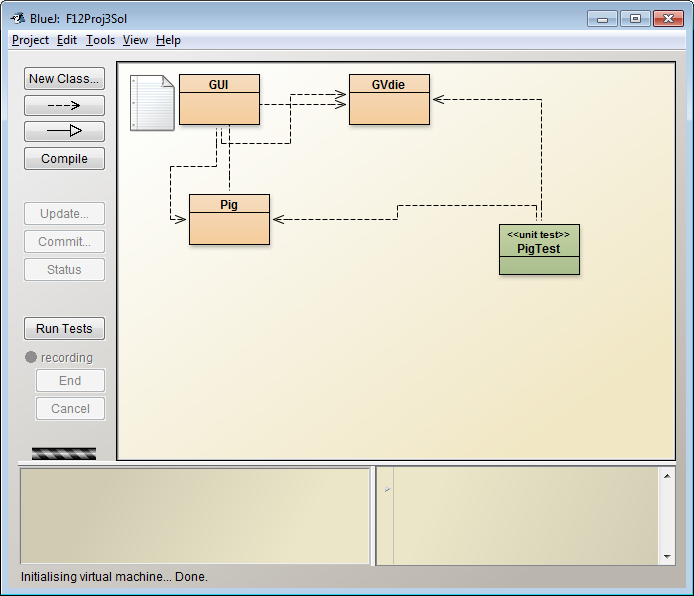
**Sample GUI**

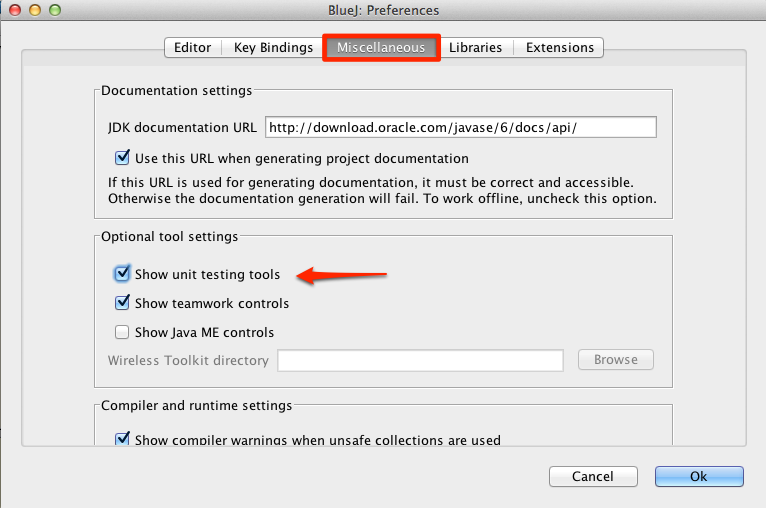


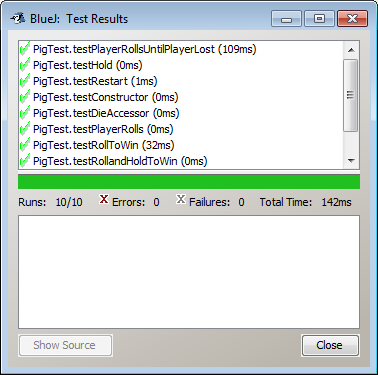
# JUnit Testing Details

JUnit is a Java library that provides a framework to automate software testing. A JUnit test file PigTest.java has been provided by your instructor. Use the instructions at the end of this document for additional information about JUnit Testing

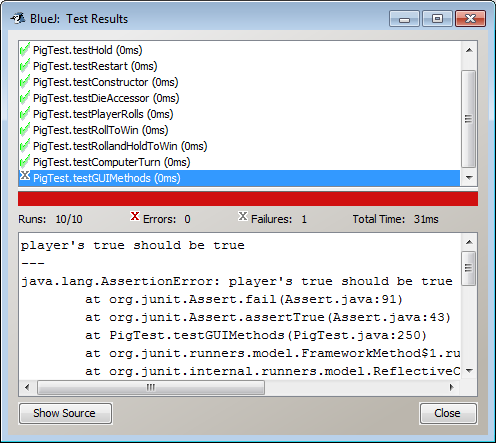
1. Name your class Pig. **Exact spelling is required**. Using a different name will cause the tests to fail.
2. Complete ALL requirements described above.
3. Download PigTest.java to the same folder as your Pig.java
4. Restart Bluej (if PigTest does not show up on your BlueJ window). BlueJ should recognize PigTest as a “<<unit test>>” file.



1. If the “Run Tests” button is not visible, go to Bluej Preferences. Under the “Miscellaneous” tab check the “Show unit testing tools”  
     
   
2. On the BlueJ window, click the “Run Tests” button. If your class passes all the test cases, the test results window will show a green horizontal bar and zero failures.



Otherwise, 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.



Click on the failed test result to see a longer description of the error below.

**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. (90 pts)
* Elegant source code that follows the GVSU [Java Style Guide](http://www.cis.gvsu.edu/studentsupport/javaguide). (10 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 are free days and the maximum late penalty is 50 pts.

**Turn In**

* A professional document **is stapled** with an attractive cover page.
* 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/academic-honesty/).
* Source code - a printout of your elegant source code for the Pig class.
* Submit on Blackboard the source code for the Pig class (Pig.java).