

CSCI-1302

Software Design
Spring 2012 – University of Georgia

Project 2

Hang Man

- Goals**
- 1) Implement inheritance, polymorphism, and interfaces in a software project
 - 2) Understand and experience the principles behind pair programming and eXtreme Programming
 - 3) Implement a client class using a predefined class, given API documentation

Points This project is worth 75 points

Due Date This project is due by 11 pm on February 21, 2012

The UML design document is due by 11 pm on February 10, 2012

Late Penalty Otherwise, 12% off the maximum original point value is deducted for each 24-hour period the assignment is late for up to two days late. Saturday/Sunday count as one day.

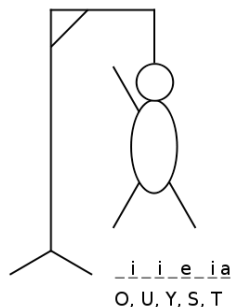
Collaboration Policy For this assignment, you may only collaborate with your assigned pair programming partner.

We expect you to abide by standard pair programming policies; all code that is written that is submitted must be written with both parties co-located at the same computer. Individuals who do not follow pair programming policies will be penalized.

The Java API, Java Tutorials, and the course textbooks are fair resources for this project.





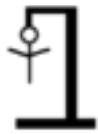
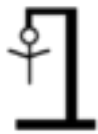


Project Introduction

In this project, pair programming teams will be randomly assigned and announced in class. Each team will be required to design and implement a hangman game.



In this project, pair programming teams will be randomly assigned and announced in class. Each team will be required to design and implement a souped-up version of the classic game, Hangman. Hangman is traditionally a pen-and-paper game where one player thinks of a word and the second person tries to guess it by suggesting letters. The word to guess is represented by “blanks” where the number of blanks is equal to the number of letters in the word. If the guessing player correctly guesses a letter in the word, the other player will write all occurrences of that letter in the word. If the guessed letter is not in the word, the other player will draw one element in the hangman diagram, which is a rudimentary image depicting a hanging man. An example image is shown on the left, an image that is part of the public domain, designed by Kyle Siehl.

Let's take a look at a sample run (which is excerpted from the Wikipedia entry on "Hangman", found at [http://en.wikipedia.org/wiki/Hangman_\(game\)](http://en.wikipedia.org/wiki/Hangman_(game)))

Word: _ _ _ _ _ Guess: E Misses:	
Word: _ _ _ _ _ Guess: T Misses: e	
Word: _ _ _ _ _ Guess: A Misses: e, t	
Word: _ A _ _ _ A _ Guess: O Misses: e, t	
Word: _ A _ _ _ A _ Guess: I Misses: e, o, t	
Word: _ A _ _ _ A _ Guess: N Misses: e, i, o, t	
Word: _ A N _ _ A N Guess: S Misses: e, i, o, t	
Word: _ A N _ _ A N Guess: H Misses: e, i, o, s, t	

Word: H A N _ _ A N Guess: R Misses: e,i,o,s,t	
Word: H A N _ _ A N Guess: Misses: e,i,o,r,s,t	
GUESSER LOSES 😞	

The game ends when either 1) the guesser has exhausted the number of misses (6: head, torso, right arm, left arm, right leg, left leg) or 2) correctly guesses the word

You will implement capabilities for both **human and computer players**; that is, you can have one of the following combinations:

Computer (guesser) Human (hangman operator)
Human (guesser) Human (hangman operator)

Each team must design and implement the following classes named as they are stated below. You may choose to add more classes, if necessary:

In the case of an error, a human player should be prompted again to enter an open spot to mark. Your team's program **must not crash or have any undesirable behavior** when processing input.

Furthermore, three types of computer players must be programmed: **NaiveComputerPlayer**, **RandomComputerPlayer**, and **CutThroatComputerPlayer**. The **NaiveComputerPlayer** will guess the first letter, alphabetically, that has not been already guessed or played. The **RandomComputerPlay** will randomly choose a letter not already guessed. The **CutThroatComputerPlayer** will use the provided guessing code to make an educated guess.

Each team must design and implement all of the following classes named as they are stated below. Your team may choose to add more classes if necessary.

1. **Board** – An interface class that defines operations of a game board
2. **HangmanBoard** – A class the implements the **Board** interface to hold a hangman board. A board can be represented by using an array if you desire.
3. **Player** – A class that represents a player of the game
4. **HumanPlayer** – A subclass of **Player** that represents a human player defined above
5. **ComputerPlayer** – A subclass of **Player** that represents a computer player defined above
6. **NaiveComputerPlayer** – A subclass of **ComputerPlayer** defined above
7. **RandomComputerPlayer** – A subclass of **ComputerPlayer** defined above
8. **CutThroatComputerPlayer** – A subclass of **ComputerPlayer** defined above

9. Hangman – A class that contains the main method to run a game of tic-tac-toe

Input and Output requirements

The Hangman class must have a main method that can process two input arguments:

- The first input argument is Operator and the second input argument is Guesser.
- The possible value for the first argument will always be computer.
 - The computer will select a word from the dictionary file and display the appropriate number of blanks, a list of the letters guessed, and anything else that is appropriate.
- The possible value for the second argument (Guesser) are human, naive, random, and cutthroat representing a player from the computer class.
- arg[3] is optional, and this is a Boolean flag to log the action to a file called logged-gameplay.output. This is overwritten for each time.
- Your team's program must support all player combinations i.e. human vs. human, naive vs.

```
java Hangman computer human
java Hangman computer naive
java Hangman computer cutthroat
java Hangman computer cutthroat
```

At the start of the game, you must display "Hangman Game" and an empty hangman board using print statements as shown in example 1. DO NOT USE ANY GUI COMPONENTS TO DRAW THE BOARD! Notice that "|" is used as a vertical separator, "-----" are used as horizontal separators.

For the player's turn, prompt the player (whether it is a human or computer) for input, process the input, and display an updated board if the input is valid. If a human player picks an invalid letter on the board, show the error message that is in example 1 for the corresponding player and re-prompt the player. Implement the input and output exactly as shown in example 1.

At the end of the game:

```
If Player X wins, print out the following
    Game Over! You guessed the word!
If Player O wins, print out the following
    Game Over! You died :-( The word was ...
```

Points

This project is worth 75 points towards your course grade. Grading of this programming project will use the following rubric:

<u>Proper documentation</u> (pre & post statements, commenting conditionals, not excessive commenting)	5 points
<u>Design Requirements & Reflection Documents</u> Includes thoroughness of the design requirements and planning of the code, as well as the reflection of the initial design specifications after the code was implemented	10 points
<u>UML Diagram</u>	10 points
<u>Human Player</u> Passing test cases and properly handling "interesting" input	10 points

<u>Computer Player: Random</u>	10 points
Passing test cases and properly handling “interesting” input	
<u>Computer Player: Naive</u>	10 points
Passing test cases and properly handling “interesting” input	
<u>Computer Player: Cutthroat</u>	10 points
Passing test cases and properly handling “interesting” input	
<u>Pair Programming Review</u>	10 points
Total:	75 points

Note: *This is our approximate grading distribution. Point values may vary.*

Extra Credit

- 1) Download and use JUnit for this project. By “use”, we mean establish test cases and make sure they pass as you develop each method. Your deliverable for this will be including all the source code for test cases, and a screenshot showing the JUnit dialog box / commandline successfully running all your test cases. This is worth up to 5 points of extra credit.
- 2) Implement your own guessing code. If your code is quicker than Doc’s, you’ll gain an additional bonus. This is worth up to 5 points of extra credit.

Submission Instructions

One project should be submitted per team

1. Create a folder in an Odin account called **lastname1_lastname2_proj2** where lastname1 and lastname2 are the two different last names of the team members.
2. Copy all **thoroughly commented** Java source files in the folder created in step 1.
3. Place a working makefile in the folder created in step 1 that has three directives:
 - a. `compile`: compiles all of the source code
 - b. `run`: runs an example of your program
 - c. `clean`: removes all class files
4. Add a `readme` file to the folder created in step 1 which has your name, your partner’s name and clear instructions on how to compile and run your team’s program.
5. Remove all class files before submitting.
6. Navigate to the parent directory of the folder created in step 1 on Odin, and issue the command below.


```
submit lastname1_lastname2_proj2 cs1302a
```
7. If the submission was successful, then a file that begins with `rec` will be created in the submitted folder.