Project 13

CS 1323, Fall 2015

# Learning Objectives

1. Use at least four distinct ArrayList objects in a single program. (10 points)
2. Use at least two methods from the Collections class. (10 points)
3. Pass different ArrayList objects to the same parameter in a method. (20 points)
4. Write a method that can be used repeatedly to alternate turns in a game by changing arguments. (20 points)
5. Use TODO: tags in eclipse to track progress. (10 points)
6. Implement the logic of the card game Go Fish! correctly. (20 points)

10 points will be awarded for the documentation of your program. That means using good names for variables, proper and consistent indentation of code, sufficient comments and meaningful use of whitespace.

Section 10: When your program is completed and running, have the teaching assistants check it to get credit for the lab. If you do not complete the laboratory during the allotted time, you may submit it on Janux before Monday, November 23 at 11:59 p.m. Only people who attend the whole laboratory will be permitted to submit assignments on Janux.

Sections 10 and 995: Submit your finished project on Janux by Monday, November 23 at 11:59 p.m.

The deadline is after the midterm examination. It will be highly beneficial for you to complete the laboratory before the examination.

I have provided sample code to get you off to a good start and help you pick methods in such a way that you can meet the learning objectives of the project.

# Description of Game

We’ll implement the card game “Go Fish!” with a human playing against a computer. The rules are below, taken from <http://en.wikipedia.org/wiki/Go_Fish> but edited significantly. Please pay attention to the rules I have here—the real game gets fairly sticky to program and this one is more simple.

A regular deck of cards consists of 52 cards. There are four suits and thirteen card ranks (Ace, 2, 3,…10, Jack, Queen, and King). We’re going to simplify our cards. The cards will have ranks from 1 to 13, and each rank will have identical cards. This removes suit from the game.

The computer deals seven cards to the human and the computer from a shuffled deck. The remaining cards are shared in a pile.

The human player should play first. The human asks the computer for all its card(s) of a particular rank that is already in his or her hand. For example Alice may ask, "Computer, do you have any threes?" Alice must have at least one card of the rank she requested in her hand. The computer must hand over all cards of that rank. If the computer has no cards of that rank, Alice is told to "Go fish," and she draws a card from the pool and places it in her own hand. When any player at any time has all four cards of one rank, it forms a book, and the cards must be removed from the hand and placed face up in front of that player.

If the player has no cards in their hand, they may not request cards form the other player, they just draw a card.

When the pile is empty, no cards are drawn, but the player still gets to ask for cards following the same rules.

The computer is not allowed to examine or deduce the human player’s cards while playing the game. The computer should randomly pick one card from their hand to request. This means that the computer is not being strategic at all and will probably lose most of the time (unless the player really stinks at Go Fish!).

When all sets of cards have been laid down, the game ends. The player with the most cards in piles wins.

The game is easier to play if the cards are printed out in sorted order. This also uses a method in the Collections class, which meets a learning objective.

If you’re enjoying working on this project, you could try coming up with a better strategy for the computer. For example, simply going through the cards one at a time is a better strategy. Asking for the card that was most recently drawn is another strategy. Don’t be afraid to have fun programming. ☺

# Implementation

There is a trick that greatly simplifies building a deck of cards. Create the deck by using a loop that counts from 0 to 51 (inclusive). The rank is determined by taking the card number % 13 + 1. This particular game does not require any understanding of suit, but if you wanted to find the suit you would use number % 4 + 1 (if the suits were unit indexed like the ranks were). A single while loop can create all 52 cards.

Your deck of cards should be an ArrayList<Integer>.

Shuffle the cards using a method in the Collections class.

The real challenge to this program is to handle multiple ArrayList objects. You will need one ArrayList for each hand, one for the pool, and one for each player’s pile. The cards (which are just Integers) should only be constructed once. They should be moved around between the various ArrayList objects as play progresses. The methods are designed to alternate between being used with the computer doing the choosing and the player doing the choosing. Learning to manipulate these methods properly is the main purpose of this laboratory.

# Warning on ArrayList<Integer>

There is one really tricky thing that happens in Java in this project. The remove method has two versions with one parmeter: one takes an int argument, and the other an Object. The one with the int argument removes an element with a given index. The one with the Object argument removes an element that is .equals() to the given object. Unfortunately, this confuses Java (with autoboxing, that we will discuss soon). It will always think you are using an index unless you use an Integer object as a parameter. The fix is shown in the code below.

ArrayList<Integer> list = new ArrayList<Integer>();

list.add(1); // 1 gets autoboxed into an Integer object

list.add(2); // 2 gets autoboxed into an Integer object

list.remove(0); // removes the value at index zero

If you want to remove the number 2, you have to make it an Integer object.

list.remove(new Integer(2)); // removes the first value 2 (starting from index 0)

// that it finds in the ArrayList—this value is at index 0

# TODO in Eclipse

Eclipse has a nice feature that we’ll use in this project. When you make a comment with TODO: (and you must have the colon), it will be put into the task list when the file is saved.

To display the Task List go to Window -> Show View -> Tasks. The Task window will show up at the bottom of the usual eclipse frames along with the Console window. Remember, it only updates when the file is saved, so don’t expect immediate responses.

These tags are used to keep track of things that need to be done. I’ve put them throughout the code to help you find the opportunities to meet the learning objectives.

This is a shortcut that every professional programmer uses. You’ll notice that eclipse puts little blue squares in the right hand margin and blue check boxes in the left hand margin to show you where the TODO: tags are located.

# Strategy

I recommend that you do some easy items before moving on to playOneTurn and playOneGame, as these will be the most challenging methods to write. Some of the TODO: items are just a single line of code.

You may want to look at the Javadoc for the class. To generate it select Project -> Generate Javadoc. If eclipse does not know where the javadoc command is, browse for it in the folder where we installed Java during the first lab of the semester. With Javadoc, when you hover over the name of the commands (after you open the GoFish file in the default package), it will show the documentation on the Javadoc tab that is next to the Tasks tab at the lower part of your screen.