CS 458 - Homework 3

Deadline
Problems 2 onward are due by 11:59 pm on Thursday, September 24, 2015
(Problem 1 was due by the end of the Week 4 Lab.)

How to submit
Submit your files for Problem 2 onward for this homework using ~st10/458submit on nrs-labs, with a homework number of 3
(Problem 1 was completed during the Week 4 Lab.)

Purpose
To make sure you know at least some git command-line-level basics, to think about some of the Extreme Programming practices, and to very lightly try out JUnit

Important notes
• Note that some of your submissions for this assignment may be posted to the course Moodle site.
• Create a file named 458hw3.txt that starts with your name. Then give the problem number and part letter for each of your answer(s) for Problems 2 and 3. (Problem 4 results in separate files to submit.)

Problem 1
Your team's lab4-option-choice.txt file from the Week 4 Lab, submitted during that lab using ~st10/458submit with a lab number of 84, is "counting" as Problem 1 of this homework.
(That is, you will be graded on if your team successfully completed this file, meeting the posted specifications, as part of your team meeting during that lab.)

Problem 2
To help encourage everyone to get "up to speed" on git, answer each of the following questions. (Remember, you have your "GitHub Git Cheat Sheet" handout from the Week 3 Lab -- which is also posted on the public course web site -- as well as that lab's projected notes and a variety of resources from github.com.)

2 part a
Assume that, at some point in the past, you have cloned your team's private repository on GitHub into a local repository on your own computer.
Give the command you can now type to get the newest updates from the team repository on GitHub, (changing your local copy).

2 part b
Assume that you have modified a file in your local repository named action.cpp so that it now implements a user story Add widget control. Give the command you can now type to stage your modified action.cpp for a commit.

2 part c
Give the command you can now type to commit, including an attempt at a descriptive log message.

2 part d
Finally, give the command you can now type to push your committed changes to the team's private repository on GitHub.

2 part e
Give the command you can use at any time to check the current status of your repository, to see what, if anything, has been modified.

Problem 3
Consider the following list of some Extreme Programming practices:

• Pair programming
• The planning game/release planning
• Unit testing/test-driven development/test-first development (assume this includes the concept of writing tests BEFORE writing the code to be tested, and assume that, once written, code cannot be checked-in to the current project version unless it passes its unit tests)
• Acceptance testing/customer testing
• Frequent releases/Small releases
• Refactoring
• Simple design/YAGNI (You Aren't Going to Need It)/"do the simplest thing that could possibly work"
• Collective code ownership
• Continuous integration
• On-site customer/"extreme" customer involvement
• 40-hour week/sustainable pace
• Coding standards
• System metaphor/common vocabulary
• User stories
• Stand-up meetings
• CRC cards - Class, Responsibilities, and Collaboration - for object-oriented programming
• Spike solutions

3 part a
You hopefully have noticed that some of these practices influence, or even depend, on some of the other practices.
Consider the practice of "Collective code ownership". List at least three other practices from the list above that make "Collective code ownership" more practical/feasible.

3 part b
CRC - Class, Responsibilities, and Collaboration - cards are used for brainstorming, for design, and for communication. They do assume an object-oriented approach. You'll find a number of templates on the web, but this is one simple version, based on the description and examples at:

http://www.agilemodeling.com/artifacts/crcModel.htm

An index card has the name of a class on the top of the card. On the left-hand side of the card, you list that class's responsibilities -- something that class knows or does. On the right-hand side of the card, you list collaborator(s) -- other classes that this class interacts with to fulfill its responsibilities.

As an example from the above web page, you might have a card with Customer at the top, with responsibilities listed on the left-hand-side of:
• places orders
• knows name
• knows address
• knows customer number
• knows order history

On the right-hand-side, it might have a collaborator-class listed of:
• Order

It is considered a benefit that index cards are small, and easy to move around on a desk (or perhaps a computer desktop) -- you can put cards for classes that collaborate with each other near each other, and cards that don't further apart. You can also easily throw away one or more cards and start over if one approach isn't working!

Consider a Monopoly game program. You might have a Player class in such a game.

Give an example of at least one responsibility that a Player class might have.
Give an example of at least one collaborator-class that a Player class might have to interact with to
Problem 4

Consider the posted GameDie.java and GameDieTest.java classes (posted along with the Week 4 Lab projections on the public course web site) which we used to demonstrate a little bit about TDD (Test-Driven Development) and JUnit unit testing.

You are expected to follow the coding and testing style of these given classes in completing this problem.

Also remember: you know, from the in-lecture examples, that when you have GameDie.java and GameDieTest.java in the same directory and open them at the same time in DrJava (freely downloadable from drjava.org), you can run the JUnit tests in GameDieTest.java using the "Test" button in DrJava.

4 part a

Make copies of the posted GameDie.java and GameDieTest.java classes, and add your name to the opening comment block of each (indicating that these are your modifications).

(Note: it is also okay if you'd like to remove some of the more "explanatory" comments from these posted examples you are starting from. But make sure each class and method still has an opening comment, at least giving the purpose of that class or method, and don't remove the javadoc-style comments.)

4 part b

Consider the user story: "A user can keep track of how many times a game die has been rolled."

Write appropriate tests in GameDieTest that should pass if a user can indeed do this -- ideally, do so in a test-driven-development approach, writing a test, then modifying GameDie to pass that test (and still pass the previously-existing tests), and so on.

• You need to add at least one new test method to GameDieTest; you also should modify at least one of its existing test methods.

• You need to add at least one private data field to GameDie; you also need to add at least one new method, and you need to modify at least two of its existing methods.

Submit your resulting GameDie.java and GameDieTest.java along with your 458hw3.txt.