PROGRAMMING ASSIGNMENT 2

Due: Thursday, February 5th at 11:59pm

This assignment will provide you experience working with

- Classes with inheritance
- Creating UML class diagrams
- Testing with JUnit (adding tests)
- Submission to Blackboard
    - A zipped folder named AssignmentTwo containing
        - Java files
            - Prisoner.java
                - 3 concrete base classes with different strategies inheriting Prisoner.java
                    - PrisonerNeverRats
                    - PrisonerAlwaysRats
                    - PrisonerRandomlyRats
        - UML Class Diagram for this project in PDF format
            - Include visibility, types and associations
        - In your submission comments, indicate if you would rat or not given the results

Prisoner's Dilemma

Suppose that you and one of your more friends are accused of a crime. The police immediately put the two of you in separate holding cells and start to ask you questions about the crime. They want you to make a deal to rat on your partner so that your partner gets a longer sentence in exchange for letting you go free. Of course they also offer the same deal to your partner.

What are your options? Remember, you don't know exactly what your partner will do because you're being held in separate cells.

Suppose you decide to keep quiet and not rat on your partner, but your partner decides to rat on you. This is the worst case scenario for you: there is now testimony against you leading to your extended sentence and your partner goes free because of helping the police with the case against you.

Suppose, instead, that you think your partner will rat on you, so you also decide to rat on your partner. Now the police have testimony against both you and your partner, so this outcome is bad for both of you.

Finally, what if you and your partner both keep quiet? The police don't have testimony against either of you, so you both get pretty short sentences.

We can represent the outcomes of you and your partner's decision-making processes by the following matrix:

		Partner	
		Defect (rat)	Criminals Cooperate
You	Defect (rat)	4 years each	You: free Partner: 6 years
	Criminals Cooperate	You: 6 years Partner: free	Each serves 1 year

In the outcome matrix, the rows indicate your choice of action and the columns indicate your partner's decision. "Defect" means that you decide to rat on your partner and "Cooperate" means that you keep quiet. The numbers in the matrix cells are the amount of time you would serve.

What is the best strategy?

In the usual Prisoner's Dilemma you don't know what your partner will do. What do you think you should do in this case? Rat or cooperate with your friend? Well, you will run a simulation to determine what is the rational decision for you.

Now, the Iterated Prisoner's Dilemma is a bit different. In this case, you will know what your friend did in the hand before. You will repeat the process a number of times...say, 100 times. What happens with the three prisoner types ? For extra credit then can you think of a good strategy that takes into account *what your partner has decided to do on previous iterations*? Does this strategy lower your average jail time?

Implementation

For this assignment, you need to implement an iterated prisoner's dilemma simulation, which calculates the percentages for Table 1 based on different types of Prisoner base class. You are given the IteratedPrisoner.java and IteratedPrisonersTest.java file. You will design your program with 4 classes. Start with one abstract class: (Prisoner.java).

Prisoner.java is an *abstract class* and must contain the following abstract methods:

- public abstract getDecision() returns boolean (true means the prisoner doesn't stay loyal and rats out the other prisoner, false means s/he cooperated with the other prisoner[1])
- public abstract notifyOutcome(int score) return the outcome of the play (number of years in jail for this player).

You will implement 3 different types of concrete prisoners that inherit from Prisoner:

- PrisonerAlwaysRats – This prisoner always returns true for getDecision() and doesn't use any previous knowledge
- PrisonerNeverRats – This prisoner always returns false for getDecision() and doesn't use any previous knowledge
- PrisonerRandomlyRats – Prisoner rats sometimes. Include an additional constructor:
    - PrisonerRandomlyRats(double percentage) – Responds true percentage of the time, else returns false. Default percentage is 0.50.

Examine your results and report in your submission comments if you would rat or not given the simulation runs.

Extra Credit:

Extra credit can be earned by developing at least one additional Prisoner subclass:

- PrisonerChangeDecision

- PrisonerChangeDecision – Create a solution based on what happened in the previous iteration. For example, decide what to do if your pal ratted you out in the previous iteration. You will need to keep a new field for this class only that keeps track of the previous iterations result. Can you beat the best Prisoner solution from the required Prisoners results? Regardless, note in your comment on blackboard that you did extra credit and what was the best approach.

[1] http://personalwebs.coloradocollege.edu/~mwhitehead/courses/2013_2014/CP222/Assignments/1/1.html