CSP1150/4150 Programming Principles

Workshop 9

Title: Classes

Objectives from Unit Outline:

- demonstrate, at an introductory level, a knowledge of the vocabulary and syntax of the Java language
- implement fundamental control structures and program sub-structures in Java including classes and objects, methods and inheritance

Objectives:

- to understand and design instance variables and instance methods and incorporate them into a class
- to use such a class in a driver program

Assessment:

- This is an Assessable Workshop Tasks

The Prisoners Dilemma

This week you will write some classes for playing a famous game, the Iterated Prisoner's Dilemma.

The Prisoner's Dilemma goes like this:

- Players simultaneously choose a move: 'c' or 'd'
- If both choose 'c', they get $3 each
- If both choose 'd', they get $1 each
- If one chooses 'c' and one chooses 'd', the 'd' player gets $5 and the 'c' player gets nothing

In the Iterated Prisoner's Dilemma (IPD), the two players play a number of rounds of Prisoner's Dilemma. The aim is to make the most money.

Rather than having all the code for this project in one class, we are going to use a number of classes, which cooperate to solve the problem. This promotes code re-use, and simplifies design, coding, testing and maintenance.

The design for this program uses 6 classes (you have to write one):

1. `IPDTest` – this is the driver class that contains the `main()` method, which provides the overall logic of the program.
2. `IPD` – this is a utility class that represents how a game of IPD works.

3. `TitForTat` – this class will represent a particular strategy for playing IPD (that is, it is what determines which moves to play) – this one plays 'c' first, then copies its opponent's last move.
4. `SuspiciousTitForTat` – this is like `TitForTat`, but plays 'd' first.
5. `Grim` – this is another strategy – it plays 'c' as long as the opponent plays 'c', but if the opponent ever plays 'd', it plays 'd' forever.
6. `Pavlov` – starts with 'c', and after that plays 'c' if both players did the same thing on the previous move, or 'd' otherwise.
7. `Human` – this class is to allow a human to play. It uses a Scanner to get the human player's moves.

On Blackboard, you will find the source code for all these classes, except for `Pavlov`. Your job is to

1. complete the `IPD` class, and
2. correctly implement `Pavlov` and
3. collect some data by playing the different players against each other.

Download and unzip the starter project from Blackboard now.

Examine the source file for `IPDTest`. You will see that `IPDTest` has some private fields, including these:

```
     /** This is the first player */
    //private static HumanPlayer player0 = new HumanPlayer();
    private static TitForTat player0 = new TitForTat();
    //private static SuspiciousTitForTat player0 = new SuspiciousTitForTat();
    //private static Grim player0 = new Grim();
    //private static Pavlov player0 = new Pavlov();

    /** This is the second player */
    //private static HumanPlayer player1 = new HumanPlayer();
    //private static TitForTat player1 = new TitForTat();
    //private static SuspiciousTitForTat player1 = new SuspiciousTitForTat();
    private static Grim player1 = new Grim();
    //private static Pavlov player1 = new Pavlov();
```

The /** comments */ describe what these fields are for. The test program will use two different players, which are created by calling their constructor, as in

```
        new TitForTat()
```

and

```
        new Grim()
```

Exactly one of the 5 possible choices for each player should be uncommented, before compiling and running IPDTest. In this way, different pairs of players can be tested against each other. This is not the most elegant way to do things, but works using the Java that you know so far.

In the `main()` method, there is this line:

```
            move0 = player0.getFirstMove();
```

In this line, `player0` is one of the players, and the code is calling that player's `getFirstMove()` method. The `getFirstMove()` method will return the first move for that player, according to its strategy. A little further down, the same method is called on the other play to get the other player's first move:

```
            move1 = player1.getFirstMove();
```

Just below that you will see the call

```
            move0 = player0.getNextMove(lastMove0, lastMove1);
```

Here the player's `getNextMove()` is called, this time passing in the previous move by the player and the opponent. The other player is then asked for its next move in the same way.

Next, the `main()` method calls the `IPD` class's `getScore()` method to find out what each player wins in each round:

```
            int score0 = IPD.getScore(move0, move1);
            int score1 = IPD.getScore(move1, move0);
```

To make all this work, on to your tasks:

The `IPD` class

This is a utility class that has one method, with the following header:

```
    /**
     * How many dollars for the first player?
     *
     * @param firstMove the first move
     * @param secondMove the second move
     *
     * @return the payout for the first player
     */
    public static int getScore(char firstMove, char secondMove)
```

Note that this is a static method – there is no need to create a `IPD` object to call it – the method belongs to the `IPD` class. This is similar to static methods in other utility classes like the `Math` class. Your task is to provide the body for the method. At the moment, the body just returns the value 0 no matter what the players' moves are.

Your task is to change this so that it implements the rules of the Prisoner's Dilemma game, given at the start of this document. You can then test this method in BlueJ, by right-clicking on the class, and selecting the method. In the requester that pops up, enter the possible moves for each player that you want to test (each can be either 'c' or 'd' – remember to include the single-quote marks).

The Pavlov class

This class represents an individual player using the Pavlov strategy as described above. You will need to create a new class. You might find it useful to use one of the other player classes, such as `TitForTat` as a starting point, and modify it.

Once you have completed this task, you should be able to complete the following table:

	TFT	STFT	Grim	Pavlov
TFT	30			
STFT			13	
Grim				
Pavlov				

For example, when a TitForTat (TFT) plays against another TFT, the first player will win $30; and when a SuspiciousTitForTat (STFT) plays against a Grim, the STFT will win $0. Which player(s) has the best average payoff?

Submission

Submit your source files, in particular, Pavlov.java and IPD.java. Since you can only submit one file, you could zip up your BlueJ project folder and submit that.

You have until **midnight on Sunday** to submit your solution on Blackboard (find the submission link for the correct week under **Assessments**).

Please don’t email submissions to your tutor – emails are too easily mislaid or lost.