Module: Core Java

Session 15: Thread Practice

* This is a practice session; you will work on Multithreading Assignments
* You can discuss your doubts with the trainer

**Assignments:**

**Assignment 1 – Threading Assignment**

**Objectives:**

* Learn to use the Java threads

**Introduction**

In this assignment, you will use Java threads to implement players for a simple card game

**Task Description**

The game has 4 players and 4 piles of cards, one between each pair of players. The players draw cards from the top and discard cards to the bottom of the piles in the following pattern:

|  |  |  |
| --- | --- | --- |
| Player | Pile | |
| Draw | Discard |
| 1 | 1 | 2 |
| 2 | 2 | 3 |
| 3 | 3 | 4 |
| 4 | 4 | 1 |

The first player to collect 4 cards of the same denomination wins the game. The card deck has 32 cards: 4 of each denomination from 0 through 7. However, your program must accept denominations greater than 7, for grading purposes (for testing and grading, we will create input decks with 4 of only one denomination between 0 and 7, 3 of the other denominations between 0 and 7, and fill out the 32 card deck with denominations greater than 7, with fewer than 4 of any of those denominations). Initially each player is dealt 4 cards in round robin fashion starting from player 1, and the remainder of the cards are added to the piles, starting from Pile 1, in round robin fashion (with the first card added to each pile the first one drawn, as in a queue).

**Strategy**

So that the same player (usually) wins for the same input deck for all of you, everyone must use the same game playing strategy. The simple strategy you should use has each player preferring certain card denominations. In particular, player 1 prefers 0's and 1's, player 2 prefers 2's and 3's, player 3 prefers 4's and 5's, and player 4 prefers 6's and 7's. This means that (after drawing a card from the proper deck) a player discards any card from a non-preferred denomination, if possible. If that is not possible, then the player discards a card from the denomination with fewer cards. To ensure progress in the game, a player may not indefinitely hold a card from a non-preferred denomination. This means that you must implement some mechanism for selecting a non-preferred denomination card that ensures that any particular non-preferred card is eventually selected for discard.

For example, if player 2 has (after drawing a card from the proper deck) one 2, one 4, one 5, and a pair of 3's, then the player would discard either the 4 or the 5. If player 3 had the same set of cards, the player would discard a 2 or a 3. An example of the other case, is that if player 2 had three 2's and a pair of 3's, then the player must discard a 3.

**Implementation**

In addition to a Main class that starts the game application, reads the input deck, deals the initial hands, and creates the various threads needed to run the game, you will need to build classes for the players and for the card piles. For the various classes, some of your methods will need to be synchronized, as discussed in recitation, to prevent multiple threads from accessing shared data simultaneously.

In addition, you will need to implement a routine that handles stopping players who did not win the game. The winning player can simply exit the run procedure that started its thread execution. However, this player must notify the other player threads that they need to stop (e.g., by setting a member value in the player class via a synchronized method, that is subsequently checked by all player threads). Each thread must print a message when it finishes (e.g., "Player 1 wins and exits", "Player 2 exits", etc.). Also, drawing and discarding is an atomic action, so a player must discard after a draw, even when the player wins. This means a player always has 4 cards in the hand whenever printing the contents of a hand.

Each player must demonstrate card playing actions by printing the card drawn, the card discarded and the current hand for each action. The action must be labeled with the player number. Each player's actions should be written to a separate output file.

For example, intermediate moves should print something like the following:

**player 1 draws a 4  
player 1 discards 4   
player 1 hand 5 5 5 6**

The final move of the winner prints something like the following:

**player 3 draws a 7  
player 3 discards a 3  
player 3 hand 7 7 7 7  
player 3 wins  
player 3 exits**

The card deck is read from standard input.

An example of an initial deck looks like this:

2 3 4 0 7 1 6 5  
6 5 0 2 1 7 4 3  
3 6 1 5 4 2 0 7  
1 2 3 0 5 6 7 4

As the final output for each player at the end of the game, the player should print three lines, appropriately customized for that player, to the player's output file. The first states whether the player won or lost, the second lists the player's hand at the end of the game, and the third lists the contents of the pile the player draws from at the end of the game. The lines look like this, with required keywords in bold:

**WIN** yes (or no)  
**HAND** card1 card2 card3 card4  
**DRAWPILE** contents of draw pile, separated by spaces (e.g., 2 5 3)

Your Java program should start from a class named Main (which must have a member function named main), read test data from an input data file specified on the command line, redirected from stdin, and write its output to four output files, which are always named player1.actions, player2.actions, player3.actions and player4.actions.