

1 Introduction

As it states on the syllabus, projects are to be done on your own and as with all assignments the sharing of files and code is strictly prohibited and constitutes an act of Academic Misconduct. Furthermore, the use of any electronic medium, such as code repositories, forums, blogs, message boards, email, etc. is strictly prohibited and constitutes an act of Academic Misconduct. You are not to discuss the project with anyone other than myself. You may use the resources that are on the COSC 117 course web page for this class in MyClasses.

As usual, you will submit all your work through the MyClasses page for this class, under Project #2. Make sure you do the formatting Shift+Ctrl+F before you submit your work. **All you need to submit are the Java code files for the program. Make sure you submit all of the java files in the project.**

1.1 Commenting

This is the minimum you need to do.

- In the main program:
 - At the top of at least, your name, the date written, and a description of the program. As usual.
 - Each method should have a description of what it does. A couple lines for smaller methods but a more detailed description for larger ones. In this you should have a description of what the parameters are coming in and what the method is returning.
 - Major blocks of code should contain brief but descriptive comments to their function.
 - All variables must be commented as to their purpose.
- For each class structure:
 - At the top have at least, your name, the date written, and a description of the class.
 - Each member variable should have a description.
 - Each member method should have a description of what it does. A couple lines for smaller methods but a more detailed description for larger ones. In this you should have a description of what the parameters are coming in and what the method is returning.
 - Major blocks of code should contain brief but descriptive comments to their function.
 - All variables in the methods must be commented as to their purpose.

2 Project

For this project you have a choice on what program you write. You can either write a version of the popular kids game of Connect Four, the computer game Minesweeper, or a Five Card Draw Poker game. **You may do one and only one of the projects.**

2.1 Poker: Five Card Draw

The objective of this project is to create a program that will play the poker game of five card draw for between 2 and 6 players. In the game of five card draw each player is dealt 5 cards then each player is allowed to throw away up to three cards and have them replaced with new cards from the

top of the deck. We will not play the version where you can draw 4 cards if you have an Ace. So when the players get the initial 5 cards these are dealt in the normal manner, one card being dealt to player 1, then one to player 2, then one to player 3, and so on cycling back to player 1 for the second card of each hand and so on. When the player throws away any cards the cards are replaced all at once from the top of the deck. So if player 1 throws away 2 cards these are replaced by the top two cards from the deck.

The program should work as follows. The user is first asked how many players will be playing the game. This input should be between 2 and 6 and the program should catch any errors in the input and if there are, ask the user for another input. The program will then deal the cards to each player (this will not be shown on the screen). Then one by one each player will be asked how many cards they would like to draw. This input must be between 0 and 3 cards, again catching any errors in the input. If the number they want to draw is greater than 0 the program should ask for the card positions to be thrown away, one by one. These inputs should be between 1 and 5, 1 representing the first card, 2 the second and so on. Again catch all input errors. Here there could be another error that is not simply inputting an integer between 1 and 5, look at the second run of the program below. If the user inputs a card number that they have already input the program should not replace that card again. This would be like being dealt a card and then throwing that same card away for another card, this is not legal in poker. After each player has had the chance to draw the program should print out the hands of all the players. At this point the program ends. See the two sample runs of the program below.

For this project you will be using the Card, Deck and PokerHand classes we discussed in class. For the PokerHand class we will add in several methods. First add in another PrintHand method, this one should take a single integer parameter called width and print out each card in the hand using width spaces. Notice in the sample runs below the cards in the hand printouts are all lined up, that is it makes it easier to read. The sample runs were done with a width of 4. Next write a replace method for the PokerHand class, the header should look like

```
public void replace(int pos, Card card)
```

where pos is the position to be replaced and card is the new card to be put into the hand. The method should make sure that pos is between 1 and 5 before it inserts the new card into the hand. Test these methods to make sure that they are working correctly before you move onto the main program.

Now for the main program. The above discussion gives you an outline of how the program is to work, here are some guidelines and hints. First, the players should be stored as an array of PokerHands, this array size will depend on the number of players of the game.

When you ask each user for the card numbers of the cards they want to replace you need to track the card numbers they have already input so that you can determine if they input the same number more than once. Personally, I would do this in an array of integers. Since the valid card positions are 1-5, I would probably fill the array with 0 and when a legitimate selection was input, place it in the array. This way, when a position was input by the user, we would first search the array for that number and if the number was not there we would add it in the array and do the card replacement. If, on the other hand, the position was in the array then I know that the user input a duplicate and hence I need to ask for the input again.

For example,

- Say the user wanted to replace three cards. We start with an array of three 0's,

0	0	0
---	---	---

- Say the user inputs a 1 when asked for a position,

Input the number of the card to replace 1-5: 1

This is a valid input, so we search the array for a 1 and since none was found we would add in the 1 into the array in the first position and our array would look like,

1	0	0
---	---	---

- Then suppose the user inputs a 3 on the next input,

Input the number of the card to replace 1-5: 3

Again, search the array for 3, it would not be found so we would put the 3 in the second position in the array, and the array would look like,

1	3	0
---	---	---

- Now say the user entered another 1,

Input the number of the card to replace 1-5: 1

Again search the array for 1, it would be found so we know that it is a duplicate, do not add in this number and do not replace position 1 again, instead, we tell the user that the position was already selected and ask the user for another number. So the array still looks like

1	3	0
---	---	---

- Then if the user inputs a 2,

Input the number of the card to replace 1-5: 2

Search the array for 2, since it is not found put a 2 in the array and proceed with the program. So the final state of the array is

1	3	2
---	---	---

At this point card numbers 1, 2, and 3 have been replaced with new cards.

2.1.1 Program Runs

Program Run
<pre> Input number of players 2-6: 3 Player 1: 2C 3C 5C 8D JC Number of cards to draw: Input the number of cards to draw 0-3: 3 Input the number of the card to replace 1-5: 1 Input the number of the card to replace 1-5: 2 Input the number of the card to replace 1-5: 3 Player 2: 2H 3S 9C JS AD Number of cards to draw: Input the number of cards to draw 0-3: 2 Input the number of the card to replace 1-5: 1 Input the number of the card to replace 1-5: 3 Player 3: 3D 4C 9H 9S KH Number of cards to draw: Input the number of cards to draw 0-3: 3 Input the number of the card to replace 1-5: 1 Input the number of the card to replace 1-5: 2 Input the number of the card to replace 1-5: 5 Player 1: 5H 5S 7H 8D JC Player 2: 3S JS QC QD AD Player 3: 6S 9H 9S JH KD </pre>

Program Run

```
Input number of players 2-6: 3

Player 1: 2H 3D 4H KD AC
Number of cards to draw: Input the number of cards to draw 0-3: 3
Input the number of the card to replace 1-5: 1
Input the number of the card to replace 1-5: 2
Input the number of the card to replace 1-5: 2
This card was already replaced, please select another card.
Input the number of the card to replace 1-5: 3

Player 2: 9D 9C 9H JD JC
Number of cards to draw: Input the number of cards to draw 0-3: 0

Player 3: 5H 8D JS QH KC
Number of cards to draw: Input the number of cards to draw 0-3: 2
Input the number of the card to replace 1-5: 1
Input the number of the card to replace 1-5: 1
This card was already replaced, please select another card.
Input the number of the card to replace 1-5: 1
This card was already replaced, please select another card.
Input the number of the card to replace 1-5: 2

Player 1: 4S 4C 5C KD AC
Player 2: 9D 9C 9H JD JC
Player 3: 2C 10S JS QH KC
```

2.1.2 Extra Credit

Once you have completed the base program above and wish to take on a little more of a challenge you may do any, or all, of the following updates you wish.

1. As you can see from the sample runs we added another feature, the hands are printed out in order by face value. This makes it easier to see what is in the player's hand. Have the display of a hand sorted from low card to high card.
2. Add in the ability to determine the type of hand the person has. A list of hand types and their descriptions are at the end of this handout.
3. Add in the ability to determine the winner of the hand. In this case you will need to be able to compare two hands to see which is better. Note that there are cases in Poker where there is a tie. For example, two straights with the same high card would be a tie. There are, of course, many other cases where ties occur.

Some hints on the extra credit.

1. Have the display of a hand sorted from low card to high card. In the class examples there are several examples of sorting an array of integers. You will alter them to work with cards. Hence you will need a way to compare face values of two cards to see if you need to move any of them around in the array. I would suggest, adding a couple methods to the Card class. First add a method called `getWorthFacesDifferent` that takes in no parameters and returns an integer of the face value of the card with J being 11, Q as 12, K as 13 and A as 14. Also for the Card class add in the following method.

```
public boolean greater(Card card2){
    return getWorthFacesDifferent() > card2.getWorthFacesDifferent();
}
```

As we discussed in class the call for a method like this is c1.greater(c2), which will return true if the face value of card c1 is greater than the face value of card c2 and false otherwise.

For the PokerHand class add in a method called sortHand that takes no parameters and will sort the hand from smallest face value to largest face value. To do this take one of the sorting examples for integer arrays and alter it to work with cards. This is why we wrote the greater method for the card class.

2. Add in the ability to determine the type of hand the person has. A list of hand types and their descriptions are at the end of this handout. Determining the type of hand is easier than it may seem at first sight. One suggestion is to create an array with 13 cells and then for each face value of the cards in a hand increment the respective cell. So if index 0 represents a 2, index 1 a 3 and so on, we have the following examples.

• 4S	4C	5C	KD	AC	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>0</td><td>0</td><td>2</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	0	0	2	1	0	0	0	0	0	0	0	1	1
0	0	2	1	0	0	0	0	0	0	0	1	1						
• 9D	9C	9H	JD	JC	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>3</td><td>0</td><td>2</td><td>0</td><td>0</td><td>0</td></tr></table>	0	0	0	0	0	0	0	3	0	2	0	0	0
0	0	0	0	0	0	0	3	0	2	0	0	0						
• 2C	10S	JS	QH	KC	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td></tr></table>	1	0	0	0	0	0	0	0	1	1	1	1	0
1	0	0	0	0	0	0	0	1	1	1	1	0						
• 9C	10S	JS	QH	KC	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td></tr></table>	0	0	0	0	0	0	0	0	1	1	1	1	0
0	0	0	0	0	0	0	0	1	1	1	1	0						
• 5D	5C	5H	7D	QC	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>0</td><td>0</td><td>0</td><td>3</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr></table>	0	0	0	3	0	1	0	0	0	0	1	0	0
0	0	0	3	0	1	0	0	0	0	1	0	0						
• 5D	5C	5H	5S	QC	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>0</td><td>0</td><td>0</td><td>4</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr></table>	0	0	0	4	0	0	0	0	0	0	1	0	0
0	0	0	4	0	0	0	0	0	0	1	0	0						

From these examples it should be clear how you can use this counting array to determine the type of hand you have.

3. Add in the ability to determine the winner of the hand. In this case you will need to be able to compare two hands to see which is better. Note that there are cases in Poker where there is a tie. For example, two straights with the same high card would be a tie. There are, of course, many other cases where ties occur.

This is probably the most involved addition to the program. If you get the addition of the type of hand you can determine that a full house wins over two pair fairly easily, but what if two players both have a straight. Then you need to look at the high card in each case. Furthermore, in this situation the Ace could be a high card or a low card. Even more interestingly, what if two players have the same two pair, then you need to look at the “kicker”. There are numerous ways to solve this problem but one you might want to think about this as follows. Come up with a scheme that will associate each possible hand with a unique number in such a way that if one hand beats out another then the winning hand will have the larger number. If you can do this then you could create a method in the PokerHand class called `getValue` that returns a long integer with this value.

```
public long getValue()
```

Note that the maximum value of a long is 9,223,372,036,854,775,807. The most significant digit is a 9 and interestingly enough there are 9 types of poker hands. There are also six blocks of three digits that follow the 9 and your hand has only 5 cards in it. So it seems plausible that you can store all the information about the specifics of the hand in these positions.

If you incorporate all of the updates, a couple sample run would look like,

Program Run

```
Input number of players 2-6: 3

Player 1: 2C 3C 5C 8D JC --- High Card
Number of cards to draw: Input the number of cards to draw 0-3: 3
Input the number of the card to replace 1-5: 1
Input the number of the card to replace 1-5: 2
Input the number of the card to replace 1-5: 3

Player 2: 2H 3S 9C JS AD --- High Card
Number of cards to draw: Input the number of cards to draw 0-3: 2
Input the number of the card to replace 1-5: 1
Input the number of the card to replace 1-5: 3

Player 3: 3D 4C 9H 9S KH --- One Pair
Number of cards to draw: Input the number of cards to draw 0-3: 3
Input the number of the card to replace 1-5: 1
Input the number of the card to replace 1-5: 2
Input the number of the card to replace 1-5: 5

Player 1: 5H 5S 7H 8D JC --- One Pair
Player 2: 3S JS QC QD AD --- One Pair
Player 3: 6S 9H 9S JH KD --- One Pair
Player 2 won this hand.
```

Program Run

```
Input number of players 2-6: 3

Player 1: 2H 3D 4H KD AC --- High Card
Number of cards to draw: Input the number of cards to draw 0-3: 3
Input the number of the card to replace 1-5: 1
Input the number of the card to replace 1-5: 2
Input the number of the card to replace 1-5: 3

Player 2: 9D 9C 9H JD JC --- Full House
Number of cards to draw: Input the number of cards to draw 0-3: 0

Player 3: 5H 8D JS QH KC --- High Card
Number of cards to draw: Input the number of cards to draw 0-3: 2
Input the number of the card to replace 1-5: 1
Input the number of the card to replace 1-5: 2

Player 1: 4S 4C 5C KD AC --- One Pair
Player 2: 9D 9C 9H JD JC --- Full House
Player 3: 2C 10S JS QH KC --- High Card
Player 2 won this hand.
```

2.2 Minesweeper

Minesweeper is a single-player logic game that has shipped with Windows system computers for years. It has become so popular that there are actually Minesweeper tournaments and world rankings for enthusiasts. The game has many variations but the original game had three levels, Beginner, Intermediate and Expert. The beginner level is pictured in Figure 1, it had an 8×8 grid of cells that were covered. There are 10 randomly placed mines in the grid and in the spots where there are not any mines there is either a blank or a number. The number represents the number of mines that are around that cell. For example, if you look at the right-hand image in Figure 1, take the 8 cells that are around each number and count the number of mines in those 8 cells and you get the number. In each blank cell there are no mines in the 8 cells around it, so instead of putting a 0 in these cells the designers of the game decided to simply leave them blank.

The player clicks on a cell and its contents are uncovered. If the player clicks on a mine the game is over, and they lose. The object of the game is for the player to uncover all of the cells that do not have any mines in them. If the player does that, they win. The second goal of the game is to do this as quickly as possible so in the upper right corner of the game is a timer.

Your program should simulate this game by allowing the player to select a row and column for the cell they wish to uncover. After each selection the program should display the updated game board and check if the player won or lost. If the player loses, and gets blown up, the game board should display all of the mines, as in the example in Figure 1. If the player wins, then the program should display a message that they won along with the total time of game play. Game play time starts when the user makes their first selection (not when the program is started) and ends when the game is over. If the player selects a cell that has a number in it then just that cell is uncovered. If the player selects a blank cell then there are no mines around it, have the program uncover the blank cell and the 8 cells that are around it. If you have played Minesweeper you know that if a player selects a blank cell then the entire blank region and its border are uncovered. Your program need not do this. Your program should, of course, check for input errors. The user must input integers between 1 and 8 for both the row and column numbers and they cannot select a cell that has already been uncovered. To track the time you can use the `System.nanoTime()` function as follows.

```
long startTime = System.nanoTime();

<< Do some process >>

long endTime = System.nanoTime();
System.out.printf("Time: %13.9f sec.\n", (endTime - startTime) / 1000000000.0);
```

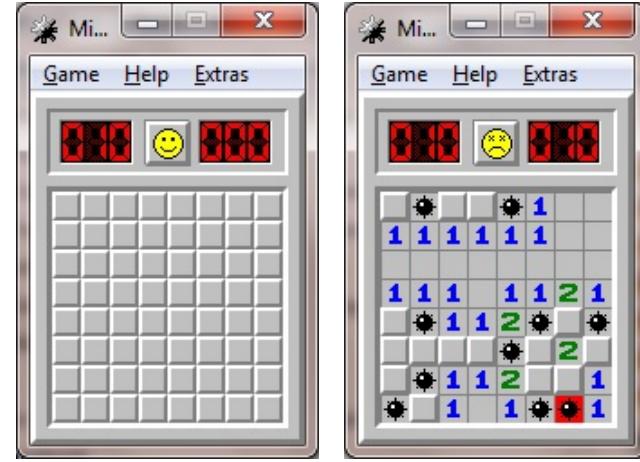


Figure 1: The Minesweeper Game

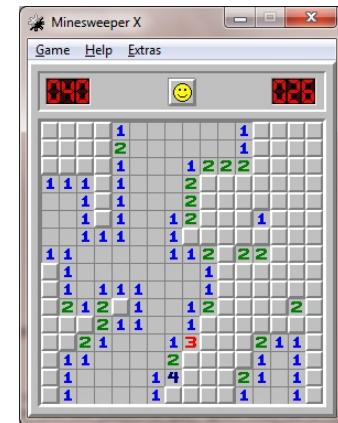


Figure 2: The Minesweeper Game: Intermediate Mode

Two example program runs are below. In these examples, I use a Q for the mines since they look a little like a bomb (use your imagination). I also use - to designate a cell that has not been

selected yet. This way the player can tell the difference between an unselected cell and a blank cell.

Program Run	
<pre> 1 2 3 4 5 6 7 8 -----</pre> <pre>1 - - - - - - - - 2 - - - - - - - - 3 - - - - - - - - 4 - - - - - - - - 5 - - - - - - - - 6 - - - - - - - - 7 - - - - - - - - 8 - - - - - - - - </pre> <pre>Input row (1-8): 2 Input column (1-8): 4</pre>	<pre> 1 2 3 4 5 6 7 8 -----</pre> <pre>1 - - - - - - - - 2 - - - 1 - - - - 3 - - - - - - - - 4 - - - - - - - - 5 1 1 - - - - - - 6 - - - - - - - 2 7 - - - - - - - - 8 - - - - - - - 1 </pre> <pre>Input row (1-8): 3 Input column (1-8): 8</pre>
<pre> 1 2 3 4 5 6 7 8 -----</pre> <pre>1 - - - - - - - - 2 - - - 1 - - - - 3 - - - - - - - - 4 - - - - - - - - 5 - - - - - - - - 6 - - - - - - - - 7 - - - - - - - - 8 - - - - - - - - </pre> <pre>Input row (1-8): 6 Input column (1-8): 8</pre>	<pre> 1 2 3 4 5 6 7 8 -----</pre> <pre>1 - - - - - - - - 2 - - - 1 - - 1 1 3 - 1 - - - - 1 - 4 - 1 - - - - 2 1 5 1 1 - - - - - - 6 - - - - - - - 2 7 - - - - - - - - 8 - - - - - - - 1 </pre> <pre>Input row (1-8): 1 Input column (1-8): 4</pre>
<pre> 1 2 3 4 5 6 7 8 -----</pre> <pre>1 - - - - - - - - 2 - - - 1 - - - - 3 - - - - - - - - 4 - - - - - - - - 5 - - - - - - - - 6 - - - - - - - 2 7 - - - - - - - - 8 - - - - - - - - </pre> <pre>Input row (1-8): 4 Input column (1-8): 1</pre>	<pre> 1 2 3 4 5 6 7 8 -----</pre> <pre>1 - - 1 - - - 2 - - 2 1 - 1 1 3 - 1 - - - - 1 - 4 - 1 - - - - 2 1 5 1 1 - - - - - - 6 - - - - - - - 2 7 - - - - - - - - 8 - - - - - - - 1 </pre> <pre>Input row (1-8): 7 Input column (1-8): 7</pre>
<pre> 1 2 3 4 5 6 7 8 -----</pre> <pre>1 - - - - - - - - 2 - - - 1 - - - - 3 - 1 - - - - - - 4 - 1 - - - - - - 5 1 1 - - - - - - 6 - - - - - - - 2 7 - - - - - - - - 8 - - - - - - - - </pre> <pre>Input row (1-8): 8 Input column (1-8): 8</pre>	<pre> 1 2 3 4 5 6 7 8 -----</pre> <pre>1 - Q 1 - Q - 2 - - 2 1 - 1 1 3 - 1 Q - - - 1 - 4 - 1 - - Q Q 2 1 5 1 1 - - - - - Q 6 - Q - - - - - 2 7 - - - Q - - Q - 8 - Q - - - - - 1 </pre> <pre>Sorry, you lost.</pre>

Program Run

```
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
-----
```

1 | - | - | - | - | - | - | - |
 2 | - | - | - | - | - | - | - |
 3 | - | - | - | - | - | - | - |
 4 | - | - | - | - | - | - | - |
 5 | - | - | - | - | - | - | - |
 6 | - | - | - | - | - | - | - |
 7 | - | - | - | - | - | - | - |
 8 | - | - | - | - | - | - | - |

Input row (1-8): 2
 Input column (1-8): 4

```
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
-----
```

1 | - | - | - | - | - | - | - |
 2 | - | - | - | 1 | - | - | - |
 3 | - | - | - | - | - | - | - |
 4 | - | - | - | - | - | - | - |
 5 | - | - | - | - | - | - | - |
 6 | - | - | - | - | - | - | - |
 7 | - | - | - | - | - | - | - |
 8 | - | - | - | - | - | - | - |

Input row (1-8): 6
 Input column (1-8): 8

```
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
-----
```

1 | - | - | - | - | - | - | - |
 2 | - | - | - | 1 | - | - | - |
 3 | - | - | - | - | - | - | - |
 4 | - | - | - | - | - | - | - |
 5 | - | - | - | - | - | - | - |
 6 | - | - | - | - | - | - | 2 |
 7 | - | - | - | - | - | - | - |
 8 | - | - | - | - | - | - | - |

Input row (1-8): 4
 Input column (1-8): 1

```
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
-----
```

1 | - | - | - | - | - | - | - |
 2 | - | - | - | 1 | - | - | - |
 3 | 1 | - | - | - | - | - | - |
 4 | 1 | - | - | - | - | - | - |
 5 | 1 | 1 | - | - | - | - | - |
 6 | - | - | - | - | - | - | 2 |
 7 | - | - | - | - | - | - | - |
 8 | - | - | - | - | - | - | - |

Input row (1-8): 8
 Input column (1-8): 8

```
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
-----
```

1 | - | - | - | - | - | - | - |
 2 | - | - | - | 1 | - | - | - |
 3 | 1 | - | - | - | - | - | - |
 4 | 1 | - | - | - | - | - | - |
 5 | 1 | 1 | - | - | - | - | - |
 6 | - | - | - | - | - | - | 2 |
 7 | - | - | - | - | - | - | - |
 8 | - | - | - | - | - | - | 1 |

Input row (1-8): 3
 Input column (1-8): 8

```
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
-----
```

1 | - | - | - | - | - | - | - |
 2 | - | - | - | 1 | - | - | 1 | 1 |
 3 | 1 | - | - | - | - | 1 | |
 4 | 1 | - | - | - | - | 2 | 1 |
 5 | 1 | 1 | - | - | - | - | - |
 6 | - | - | - | - | - | - | 2 |
 7 | - | - | - | - | - | - | - |
 8 | - | - | - | - | - | - | 1 |

Input row (1-8): 1
 Input column (1-8): 4

```
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
-----
```

1 | - | - | 1 | | | - | - |
 2 | - | - | 2 | 1 | | - | 1 | 1 |
 3 | 1 | - | - | - | - | 1 | |
 4 | 1 | - | - | - | - | 2 | 1 |
 5 | 1 | 1 | - | - | - | - | - |
 6 | - | - | - | - | - | - | 2 |
 7 | - | - | - | - | - | - | - |
 8 | - | - | - | - | - | - | 1 |

Input row (1-8): 8
 Input column (1-8): 1

<<< More Selections >>>

```
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
-----
```

1 | 1 | - | 1 | | | 1 | - | 1 |
 2 | 1 | 2 | 2 | 1 | | 1 | 1 | 1 |
 3 | 1 | 1 | - | 2 | 2 | 2 | 1 | |
 4 | 1 | 1 | 1 | 2 | - | - | 2 | 1 |
 5 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | - |
 6 | 1 | - | 2 | 1 | 1 | 1 | 2 | 2 |
 7 | 2 | 2 | 3 | - | 1 | 1 | - | 1 |
 8 | 1 | - | 2 | 1 | 1 | 1 | 1 | 1 |

Congratulations, you won!
 Your time was 45.3 seconds.

In the above examples we used a game board that looks like the following, if all of the cells were uncovered.

Example Game Board								
1 2 3 4 5 6 7 8								
1	1 Q 1		1 Q 1					
2	1 2 2 1		1 1 1					
3		1 Q 2 2 2 1						
4		1 1 2 Q Q 2 1						
5	1 1 1 1 2 2 2 Q							
6	1 Q 2 1 1 1 2 2							
7	2 2 3 Q 1 1 Q 1							
8	1 Q 2 1 1 1 1 1 1							

2.2.1 Extra Credit

Once you have completed the base program above and wish to take on a little more of a challenge you may do any, or all, of the following updates you wish.

1. In the base program, if the player selects a blank cell the program will uncover the blank cell and the 8 cells that are around it. Add in the way the original Minesweeper worked by uncovering the entire blank region and its border uncovered.
2. Add in Intermediate and Expert modes that have larger grids, and ask the user at the beginning of the game which mode they would like to play. Use say a 12×12 grid for the Intermediate mode and a 16×16 grid for the Expert mode.
3. Add in the ability to play several games during the run of the program and have the computer track the number of wins and loses. When the user quits the game, have it display the number of wins and loses.

2.3 Connect Four

Connect Four is a game that has a game board similar to the one pictured in Figure 3. There are two players, your game will be a two player game with both human players, you do not need to program the computer to be one of the players as we did with Tic-Tac-Toe. One player is red and the other is black. As you can see from the image in Figure 3, the game board has 6 rows and 7 columns. When it is a player's turn they take their color disk and place it in the top slot over one of the columns. The disk then drops down to the top of the stack in that is in that column. For example, from the image in Figure 3, if the next move is black's move and they select the second column from the left the second column from the left would have black, red, black and black as the colors going from top to bottom. Furthermore, the second column from the left would still have two empty spaces at the top. A player cannot place a disk in a column that is filled, as with the third column from the right. The game ends when one player gets four of their color disks in a line, either horizontally, vertically or diagonally. For example, in Figure 3, red has won the game since there are four red disks on a diagonal line on the right half of the board.

Your program should simulate this game by allowing each player to select a column and after each selection the program should display the updated game board and check for a winner. If there is a winner the program should display who wins. The board should look similar to the example below, using B and R for black and red. Your program should, of course, check for input errors, the user must input an integer between 1 and 7 to select a column and they cannot select a column that is full. A portion of a sample run is below.



Figure 3: The Connect Four Game

Program Run

```
| | | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
-----|1|2|3|4|5|6|7|
Player 1:  
Select a column (1-7): 3  
  
| | | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | R | | | |
-----|1|2|3|4|5|6|7|
Player 1:  
Select a column (1-7): 3  
  
| | | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | R | | |
-----|1|2|3|4|5|6|7|
Player 2:  
Select a column (1-7): 6  
  
| | | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | R | |
-----|1|2|3|4|5|6|7|
Player 2:  
Select a column (1-7): 3  
  
| | | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | B | |
-----|1|2|3|4|5|6|7|
```

2.3.1 Extra Credit

Once you have completed the base program above and wish to take on a little more of a challenge you may do any, or all, of the following updates you wish.

1. Add in different game board sizes with the user having the option to select the game board size at the start of the game. Say 10 columns and 8 or 9 rows, or 15 columns and 12–14 rows. You could even let the user select the number of columns (say between 7 and 15) and the number of rows (say between 6 and 14).
2. Add in the ability for more than two players, say any number between 2–4. You will need to designate colors for the chips. You will also need to use a larger board to make the game more interesting.
3. Add in the ability to play several games during the run of the program and have the computer track the number of wins and losses for each player. When the users quit the game, have it display the final scores.

3 Grading

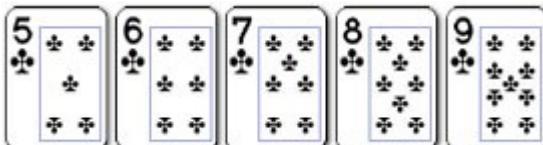
The program itself should, of course, be nicely formatted and commented and should follow all the other rules of good programming style. Use the built-in formatting tool in Eclipse and put in some vertical white space to aid in the readability but don't over do it. Variable names should be representative of their purpose. As always, there must be our standard header comment. All variables must have a comment to their use and major blocks of code should contain brief but descriptive comments to their function.

You do not need to write up a formal algorithm for this project but you should think about what needs to be done, what needs to be stored and what needs to be displayed on the screen, before you begin coding. You may also consider doing the project in stages, that is, get some portions working then add in more functionality incrementally.

The grading of the project will take two forms, a sample run and an inspection of the code. If the program does not run you will receive a zero for that portion. So even if the program is not complete you will get a better grade for a partial program that runs verses a program that does not run. So I would suggest a completion in stages approach, as I mentioned above. The run portion of the grading will test the user interface for usability and conforming to the specifications I have outlined above. The code inspection portion of the grade will involve commenting, readability, correct indentation, variable names, structure and style, correctness, and conforming to specifications.

Winning Poker Hands

Straight Flush



Five cards in sequence, of the same suit. In the event of a tie: Highest rank at the top of the sequence wins. The best possible straight flush is known as a **royal flush**, which consists of the ace, king, queen, jack and ten of a suit. A royal flush is an unbeatable hand.

Four of a Kind



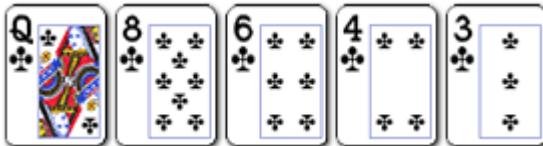
Four cards of the same rank, and one side card or 'kicker'. In the event of a tie: Highest four of a kind wins.

Full House



Three cards of the same rank, and two cards of a different, matching rank. In the event of a tie: Highest three matching cards wins.

Flush



Five cards of the same suit. In the event of a tie: The player holding the highest ranked card wins. If necessary, the second-highest, third-highest, fourth-highest, and fifth-highest cards can be used to break the tie. If all five cards are the same ranks, it is a tie. The suit itself is never used to break a tie in poker.

Straight



Any five consecutive cards of different suits. In the event of a tie: Highest ranking card at the top of the sequence wins. Note: The Ace may be used at the top or bottom of the sequence, and is the only card which can act in this manner. A,K,Q,J,T is the highest (Ace high) straight; 5,4,3,2,A is the lowest (Five high) straight.

Three of a Kind



Any three cards of the same rank. In the event of a tie: Highest ranking three of a kind wins.

Two Pair



Any two cards of the same rank together with another two cards of the same rank. In the event of a tie: Highest pair wins. If players have the same highest pair, highest second pair wins. If both players have two identical pairs, highest side card wins. If side cards are identical it is a tie.

One Pair



Two cards of a matching rank, and three unrelated side cards. In the event of a tie: Highest pair wins. If players have the same pair, the highest side card wins, and if necessary, the second-highest and third-highest side card can be used to break the tie.

High Card



Any hand that does not qualify under a category listed above. In the event of a tie: Highest card wins, and if necessary, the second-highest, third-highest, fourth-highest and smallest card can be used to break the tie.