**A. Design**

Modularity: Does 1 file occupy 1 class ONLY? 

Modularity: Do the classes contain distinct methods that have very specific functions? ****

Modularity: Where applicable, did you minimize variable and method re-declarations via inheritance?

Object-Oriented Design: Does each class have the appropriate access modifier defined (example: public, private, protected)? 

Object-Oriented Design: Does each variable have one setter (modifier) method and one getter (accessor) method in the class? 

Object-Oriented Design: Do all variables of the class have the appropriate access modifiers defined (example: public, private, protected)? 

Object-Oriented Design: Do all methods of the class have the appropriate access modifiers defined (example: public, private, protected)? 

Object-Oriented Design: When developing source code, did you minimize code writing via inheritance and polymorphism?

Efficient algorithms: Did your algorithm perform the correct calculations? 

Efficient algorithms: Where applicable, would a more efficient API or algorithm have been used to implement something in a better way?

Efficient algorithms: Did the program use API or algorithm based on the current reading materials of the week?

**B. Functionality**

Program compiles successfully. Automatic 10 points if it did not. Works

Program fulfills all functionality/All Requirements fulfilled. I am expecting 3 classes according to project requirements. Specific points for this Project based on Project specifications (1 point each):

1. Class 1 should define the GUI and should be hand-coded and not generated by a GUI generator 
2. Class 1: In addition to the main method and a constructor to build the GUI, event handlers will be needed to handle each of the four buttons shown above. When the *Withdraw* button is clicked, several checks must be made. 
3. Class 1: The main class must contain two Account objects, one for the checking account and another for the savings account. 
4. I used one object that holds both the checking and savings account balances
5. Class 1: Event handlers need to be defined to handle the radio buttons for checking account and savings account. If the radio button for checking account is selected, it will point to the Account object representing checking account. If the radio button for savings account is selected, it will point to the Account object representing savings account. 
6. I used a boolean variable to keep track of which account was selected
7. Class 1: The first check is to ensure the value in the text field is numeric. 
8. This is my second check because I decided to add a check to make sure the user had entered something into the textfeild.
9. Class 1: Next a check must be made to ensure the amount is in increments of $20. 
10. This is my fourth check because I decided to add a check to make sure that the numerical value in the textfeild is positive.
11. Class 1: At that point an attempt to withdraw the funds is made from the account selected by the radio buttons. The attempt might result in an exception being thrown for insufficient funds. 
12. Class 1: If any of those three errors occur a JOptionPane window should be displayed explaining the error. 
13. Class 1: Otherwise a window should be displayed confirming that the withdrawal has succeeded. 
14. Class 1: When the *Deposit* button is clicked the only necessary check is to ensure that the amount input in the textfield is numeric. 
15. Class 1: Clicking the *Transfer* button signifies transferring funds to the selected account from the other account. 
16. Class 1: The checks needed are to confirm that the amount supplied is numeric and that there are sufficient funds in the account from which the funds are being transferred. 
17. Class 1: Clicking the *Balance* button will cause a JOptionPane window to be displayed showing the current balance in the selected account. 
18. I show both the checking and saving account balances of the JOptionPane
19. Class 2: The second class is Account. It must have a constructor plus a method that corresponds to each of the four buttons in the GUI.
20. I decided to leave these methods within the actions listeners defined in the GUI class
21. Class 2: It must also incorporate logic to deduct a service charge of $1.50 when more than four total withdrawals are made from either account. Note that this means, for example, if two withdrawals are made from the checking and two from the savings, any withdrawal from either account thereafter incurs the service charge.
22. I incorporated this into my withdrawal button action listener in the GUI class
23. Class 2: The method that performs the withdrawals must throw an InsufficientFunds exception whenever an attempt is made to withdraw more funds than are available in the account. Note that when service charges apply, there must also be sufficient funds to pay for that charge.
24. Class 3: The third class is InsufficientFunds, which is a user defined checked exception.

Confused what this class is supposed to do

Program Outputted Correct Results

1. User interface is displayed as expected (1 point).
2. All buttons work according to specifications (2 points). 
3. Both JOptionPane windows displayed correctly (2 points). 

**Extra Effort**

Please demonstrate in your documentation with the heading EXTRA EFFORT indicating where you performed EXTRA EFFORT. EXTRA EFFORT must be a technology or algorithm outside the current study.

Examples of unsuitable or inappropriate EXTRA EFFORT activities:

1. I added an extra variable to the class
2. I added an additional line of source code.
3. I used a Java technology from a previous Chapter.

**C. Test Plan**

Test Plan should be defined in a commented documentation section on top of the main Java source file.

Because this is a user-interface application. You must demonstrate at least 4 Test Cases that satisfies the Functionality requirements of the Project. 

Yes, I am expecting some discussion on the limitations of the Test Plan. 2 bullet points/individual ideas minimum. Each bullet point must contain at least 20 words of information.

No Test Plan means 0 points for entire Test Plan section. 

**D. Documentation**

Documentation section should be defined in a commented documentation section on top of the main Java source file. 

I am expecting a Lessons Learned section in the commented documentation section with at least 2 bullet points/individual ideas. Each bullet point must contain at least 20 words of information. 

I am expecting a Possible Improvements section in the commented documentation section with at least 2 bullet points/individual ideas. Each bullet point must contain at least 20 words of information.

I am expecting a Approach section in the commented documentation section with at least 2 bullet points/individual ideas. Each bullet point must contain at least 20 words of information.

No Documentation means 0 points for entire Documentation section. 