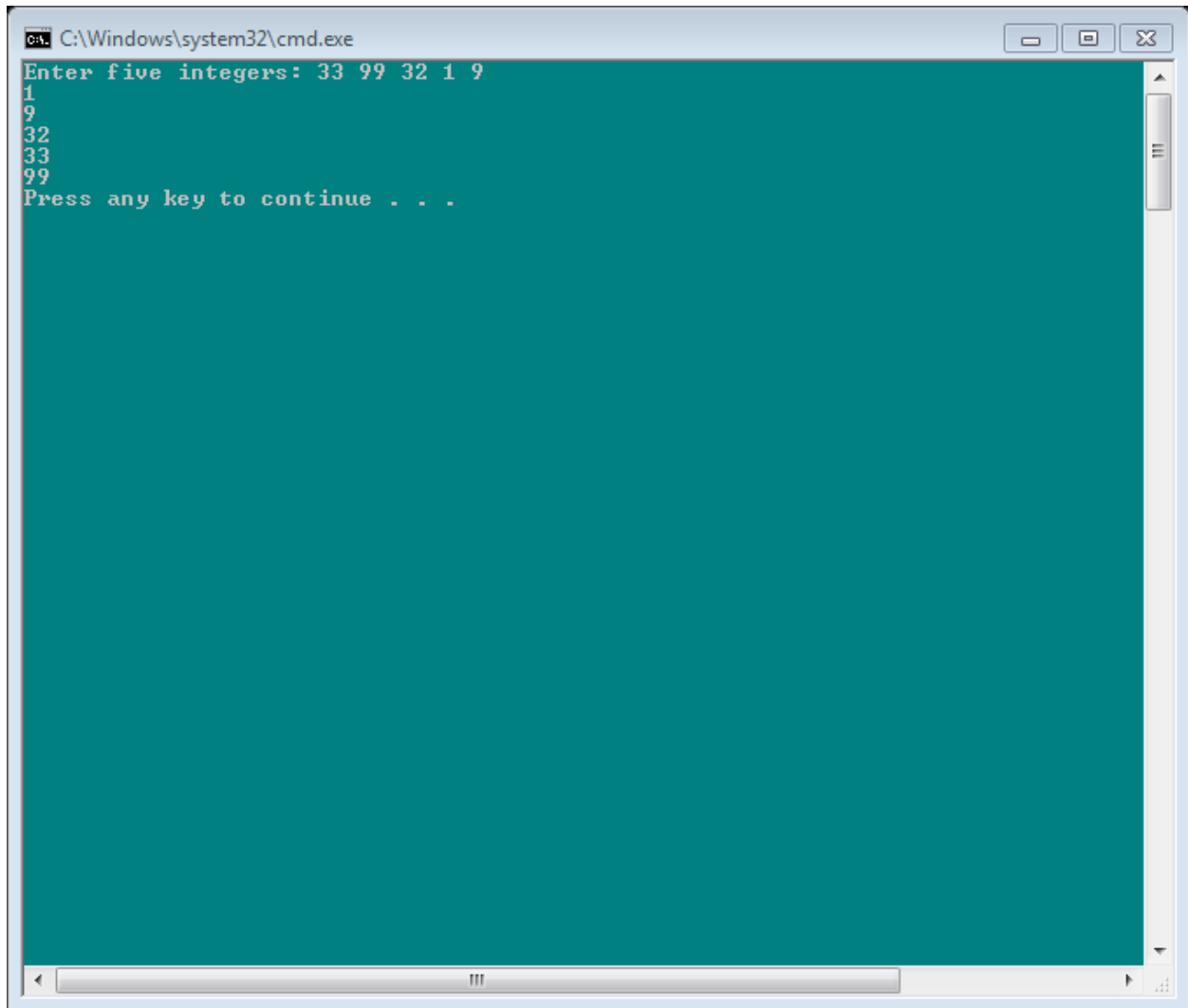


CIT 249: Java II

Chapter 11 Lab 3

In this lab we will complete #11 under Programming Exercises on page 444. When the program is ran the result will be:



```
ca. C:\Windows\system32\cmd.exe
Enter five integers: 33 99 32 1 9
1
9
32
33
99
Press any key to continue . . .
```

As shown in the screenshot, the user is prompted to enter 5 integers. Once the integers have been entered they will be sorted from lowest to highest.

1. Open a new document and save the file as Ch11Lab3.java.
2. Type the following import statements:

```
import java.util.ArrayList;
import java.util.Scanner;
```

3. Type the class header, opening brace, main method header and opening brace for that method.
4. First we will create a Scanner object to handle input and also create an ArrayList. Type:

```
Scanner input = new Scanner(System.in);
```

```
ArrayList<Integer> list = new ArrayList<Integer>();
```

- An ArrayList is similar to an array except that it is expandable during run-time, whereas an array is not.
- ArrayList are part of the Collections class, which we will cover thoroughly in a later chapter. Recent versions of the JDK require that you designate what type of content they will hold. The program will compile without the information but you would receive a warning. In the above we designate that the ArrayList will hold integers, by designating the Integer wrapper class. You cannot designate the type using int or any primitive type.

5. Next we request input from the user and add each integer to the ArrayList. Type:

```
System.out.print("Enter five integers: ");  
for (int i = 0; i < 5; i++)  
    list.add(input.nextInt());
```

- The ArrayList has a method named add() which adds items to the ArrayList.

6. We invoke the sort() method, passing the ArrayList to it, by typing:

```
sort(list);
```

7. We display the numbers in order from smallest to highest by typing:

```
for (int i = 0; i < list.size(); i++)  
    System.out.println(list.get(i) + " ");
```

8. The sort method is next, which will sort the numbers. Start by typing:

```
public static void sort(ArrayList<Integer> list) {  
    for (int i = 0; i < list.size() - 1; i++) {  
        // Find the minimum in the list[i..list.length-1]  
        int currentMin = list.get(i);  
        int currentMinIndex = i;
```

- Using a for loop we set the beginning point as when i = zero, the ending point of when i is less than the ArrayList's size minus 1, and increment by 1
- We create two integers, one for current minimum number which receives the value of the current item in the list, and the index of said number.

9. We use an inner for loop to check the number to see if they are the smallest. Type:

```
for (int j = i + 1; j < list.size(); j++) {
    if (currentMin > list.get(j)) {
        currentMin = list.get(j);
        currentMinIndex = j;
    }
}
```

10. We swap numbers where appropriate. Type:

```
// Swap list.get(i) with list.get(currentMinIndex) if necessary;
if (currentMinIndex != i) {
    list.set(currentMinIndex, list.get(i));
    list.set(i, currentMin);
}
```

- Notice that the ArrayList also has a set() method.

11. We finish by closing the outer for loop, method and class by typing:

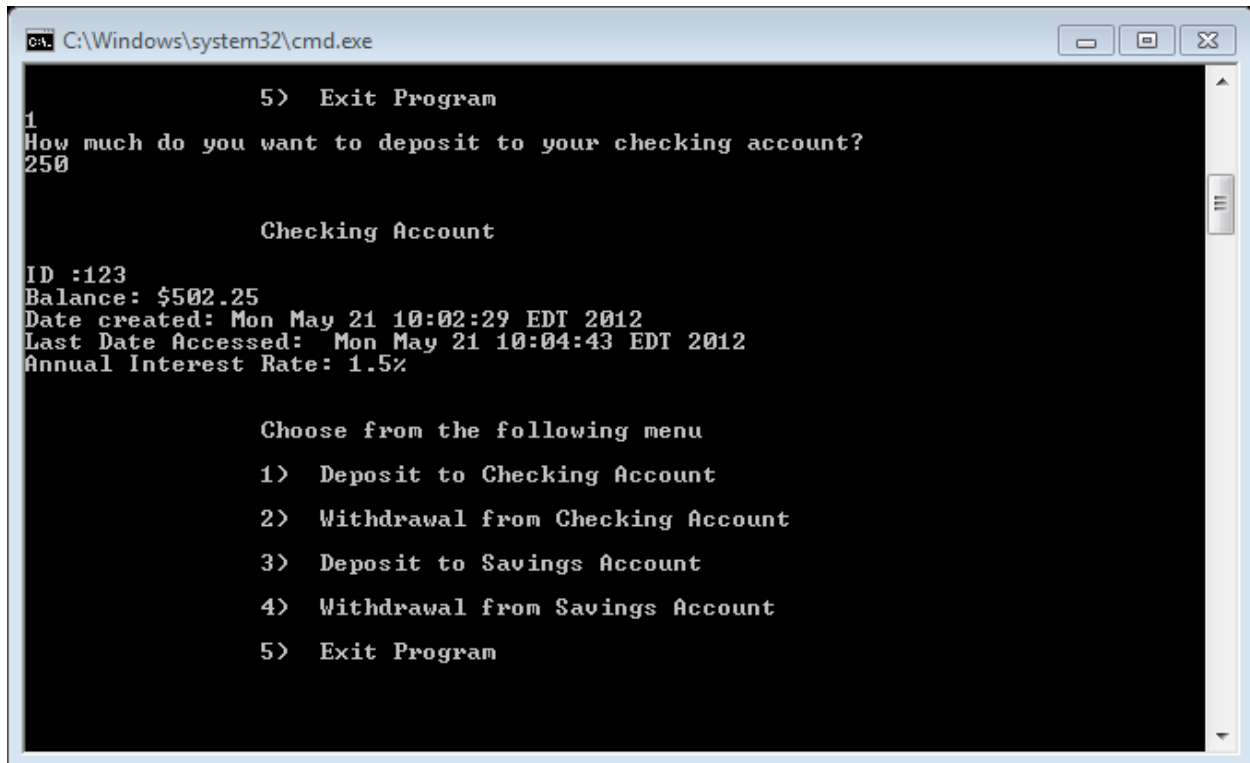
```
    }
}
}
```

12. Compile the program and fix any errors if necessary.

13. Run and test the program.

14. Compile the .java and .class file into a single zip or rar file and submit.

When the user selects 1 from the menu:



```
C:\Windows\system32\cmd.exe

                    5> Exit Program
1
How much do you want to deposit to your checking account?
250

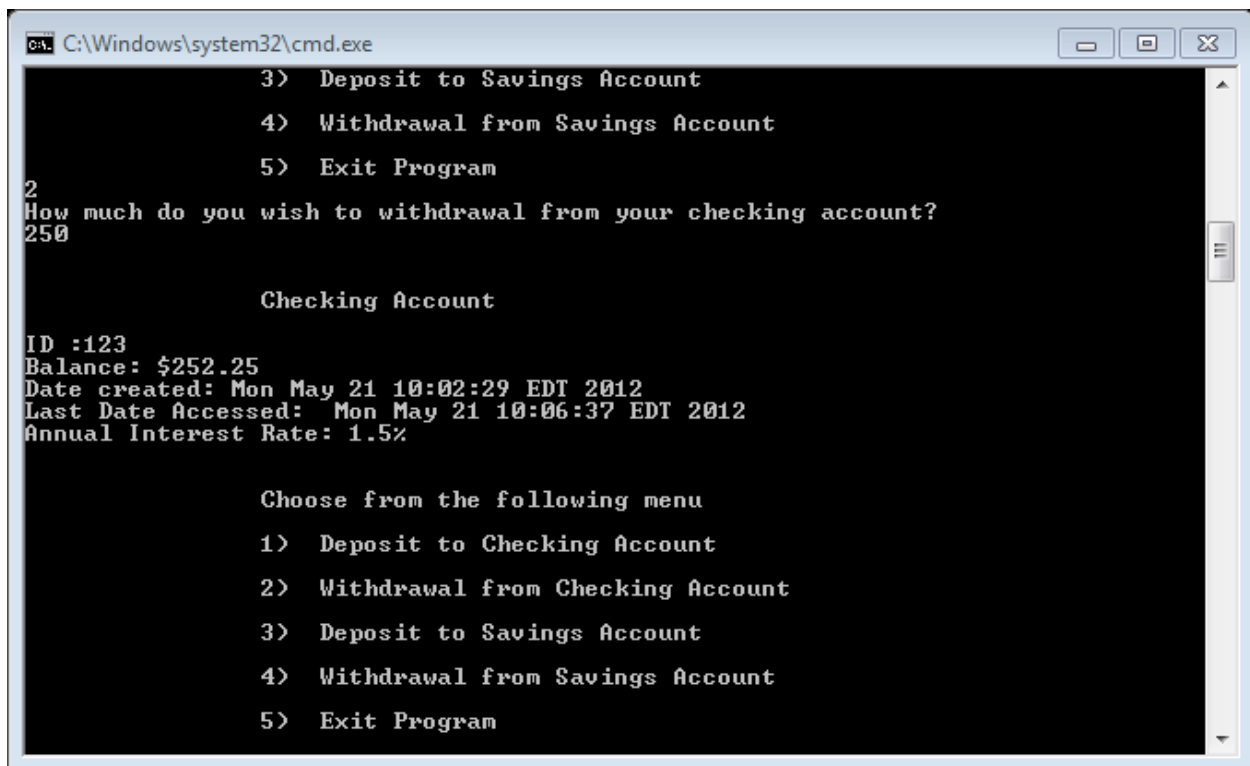
                    Checking Account

ID :123
Balance: $502.25
Date created: Mon May 21 10:02:29 EDT 2012
Last Date Accessed: Mon May 21 10:04:43 EDT 2012
Annual Interest Rate: 1.5%

                    Choose from the following menu

                    1> Deposit to Checking Account
                    2> Withdrawal from Checking Account
                    3> Deposit to Savings Account
                    4> Withdrawal from Savings Account
                    5> Exit Program
```

Notice that the menu reappears after displaying the new balance? If 2 is selected:



```
C:\Windows\system32\cmd.exe

                    3> Deposit to Savings Account
                    4> Withdrawal from Savings Account
                    5> Exit Program
2
How much do you wish to withdrawal from your checking account?
250

                    Checking Account

ID :123
Balance: $252.25
Date created: Mon May 21 10:02:29 EDT 2012
Last Date Accessed: Mon May 21 10:06:37 EDT 2012
Annual Interest Rate: 1.5%

                    Choose from the following menu

                    1> Deposit to Checking Account
                    2> Withdrawal from Checking Account
                    3> Deposit to Savings Account
                    4> Withdrawal from Savings Account
                    5> Exit Program
```

If 3 is selected:

```
C:\Windows\system32\cmd.exe

3> Deposit to Savings Account
4> Withdrawal from Savings Account
5> Exit Program
3
How much do you want to deposit to your savings account?
250

Savings Account

ID :123
Balance: $502.25
Date created: Mon May 21 10:02:29 EDT 2012
Last Date Accessed: Mon May 21 10:07:11 EDT 2012
Annual Interest Rate: 4.5%

Choose from the following menu
1> Deposit to Checking Account
2> Withdrawal from Checking Account
3> Deposit to Savings Account
4> Withdrawal from Savings Account
5> Exit Program
```

If 4 is selected:

```
C:\Windows\system32\cmd.exe

3> Deposit to Savings Account
4> Withdrawal from Savings Account
5> Exit Program
4
How much do you wish to withdrawal from your savings account?
350

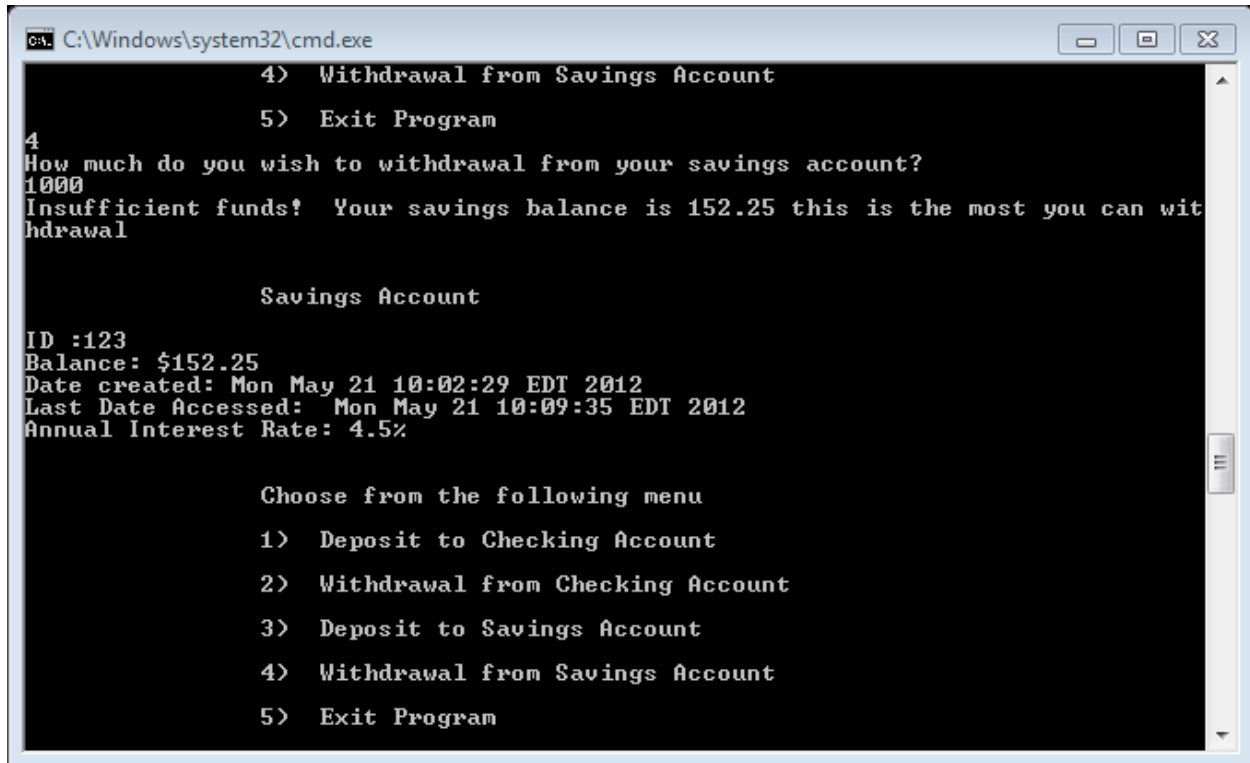
Savings Account

ID :123
Balance: $152.25
Date created: Mon May 21 10:02:29 EDT 2012
Last Date Accessed: Mon May 21 10:07:54 EDT 2012
Annual Interest Rate: 4.5%

Choose from the following menu
1> Deposit to Checking Account
2> Withdrawal from Checking Account
3> Deposit to Savings Account
4> Withdrawal from Savings Account
5> Exit Program
```

Finally if 5 is selected the program will exit.

The user is allowed to be overdrawn on the checking account but not the savings account. If they attempt to withdraw too much from the savings account they will receive the message of:



```

C:\Windows\system32\cmd.exe
4> Withdrawal from Savings Account
5> Exit Program
4
How much do you wish to withdrawal from your savings account?
1000
Insufficient funds! Your savings balance is 152.25 this is the most you can withdrawal

Savings Account
ID :123
Balance: $152.25
Date created: Mon May 21 10:02:29 EDT 2012
Last Date Accessed: Mon May 21 10:09:35 EDT 2012
Annual Interest Rate: 4.5%

Choose from the following menu
1> Deposit to Checking Account
2> Withdrawal from Checking Account
3> Deposit to Savings Account
4> Withdrawal from Savings Account
5> Exit Program

```

Notice the message of "Insufficient funds!...." The same balance prior to the attempted withdraw displays.

If the user attempts to overdraw on their checking account:

```

C:\Windows\system32\cmd.exe
3> Deposit to Savings Account
4> Withdrawal from Savings Account
5> Exit Program
2
How much do you wish to withdrawal from your checking account?
1000

Checking Account
ID :123
Balance: -$747.75
Date created: Mon May 21 10:02:29 EDT 2012
Last Date Accessed: Mon May 21 10:11:01 EDT 2012
Annual Interest Rate: 1.5%

Choose from the following menu
1> Deposit to Checking Account
2> Withdrawal from Checking Account
3> Deposit to Savings Account
4> Withdrawal from Savings Account
5> Exit Program

```

They are allowed to do so. Normally the client would be charged a fee for overdrawing the account. Try working that out for yourself. If you decide to add this to the program add a message listing the fee for the overdraft.

Let's get started!

Account class

The Account class has all the attributes that would be associated with both Checking and Savings accounts.

1. Open a new document and save the file as Account.java.
2. First we include some documentation explaining the purpose. Type:

```
/* Purpose: Set up the standard attributes for all types of banking accounts.
    Includes the set up on the initial date, interest rate, balances.
*/
```

3. Type the import statement that will import the Date class located within the java.util package.
4. Type the class header and opening brace.

5. We will declare several variables that are private. Private variables cannot be directly changed. Type:

```
private int id;  
private double balance;  
private double annualInterestRate;  
private Date dateCreated, dateAccessed;
```

- Notice that we create two Date objects, one for the date the account was created and the other for the last date accessed.

6. The only variable we will assign initial values to is the Date object for date created. This will be done within the default constructor method. Type:

```
public Account()  
{  
    dateCreated = new Date();  
}
```

7. We overload the constructor method twice with different parameters. Type:

```
public Account(int newId, double newBalance)  
{  
    id = newId;  
    balance = newBalance;  
    dateCreated = new Date();  
}
```

```
public Account(int newId, double newBalance, double interestRate)  
{  
    id = newId;  
    balance = newBalance;  
    annualInterestRate = interestRate;  
    dateCreated = new Date();  
}
```


8. We create our accessor methods. Type:

```
public int getId()  
{  
    return id;  
}
```

```
public double getBalance()  
{  
    return balance;  
}
```

```
public double getAnnualInterestRate()  
{  
    return annualInterestRate;  
}
```

```
public Date getDateCreated()  
{  
    return dateCreated;  
}
```

```
public Date getDateAccessed()  
{  
    return dateAccessed;  
}
```

- Accessor methods always have a return statement. In the method header the data type that will be returned is listed. These methods often do not have parameters.

9. Mutator methods change the current value of variables. Type:

```
public void setId(int newId)  
{  
    id = newId;  
}
```

```
public void setBalance(double newBalance)  
{  
    balance = newBalance;  
}
```

```
public void setAnnualInterestRate(double newAnnualInterestRate)
{
    annualInterestRate = newAnnualInterestRate;
}
```

```
public void setDateAccessed(Date newDate)
{
    dateAccessed= newDate;
}
```

- Mutator methods are always void. They normally have parameters, these are receiving new values from another location.
- They contain code that changes the variables to the new value passed to the method.

10. Our next method appears to be an accessor methods but it actually is not since it is not returning the value of a variable declared in the program. Type:

```
public double getMonthlyInterest()
{
    return balance * (annualInterestRate / 1200);
}
```

- Here we return the value of the monthly interest by multiplying the annual interest divided by 1200 by the value of the variable *balance*.

11. Our final two methods are for withdrawals and deposits. Type:

```
public void withdraw(double amount)
{
    balance -= amount;
}
```

```
public void deposit(double amount)
{
    balance += amount;
}
```

- Both of these methods adjust the value of the *balance* variable.

12. Close the class.

13. Compile the program. If necessary fix any errors and recompile. Do not run this program.

The main class

This class will contain the main method. It will also contain two additional classes for the Savings and Checking accounts. The main method will test these two classes.

1. Open a new document window and save the file as Chapter11Lab2.java.
2. First a brief purpose of the program. Type:

```
/* Purpose: Create two classes that extend the Account class.  
   Test both classes for setting up Savings and Checking accounts.  
*/
```

3. Several classes need to be imported. Type:

```
import java.util.Scanner;  
import java.text.DecimalFormat;  
import java.util.Date;
```

4. Type the class header and opening brace.
5. First we will declare several variables and create a Scanner object. Type:

```
//create variables, Date and Scanner objects  
int ID, choice= 1;  
double checkingBalance, savingBalance, checkingDeposit, checkingWithdrawal, savingDeposit,  
    savingWithdrawal, checkingOverage=0;  
Date dateAccessed;  
Scanner keyboard = new Scanner(System.in);
```

- Each variable name is self explanatory. Notice that a couple of variables we gave initial values to. When to give initial values to most variables is dependent upon the code where they are referenced. You cannot directly reference a variable that has not been first given a value. For example if you use if/else statements to set the value of a variable, the variable must first receive a value.

6. Next we request information from the user and apply their responses to several of our variables. Type:

```
System.out.println("\t\tAccount Set Up");
```

```
System.out.println("Enter the ID number");
ID = keyboard.nextInt();
```

```
System.out.println("Enter the initial amount of deposit in your checking account");
checkingBalance = keyboard.nextDouble();
```

```
System.out.println("Enter the initial amount of deposit in your savings account");
savingBalance = keyboard.nextDouble();
```

- On the first line I use the escape character of `\t` to indent text.

7. I create two objects, passing to the constructor methods the values of the user input. I also display the results. Type:

```
Checkings c1 = new Checkings(ID, checkingBalance);
System.out.println(c1);
```

```
Saving s1 = new Saving(ID, savingBalance);
System.out.println(s1);
```

8. Through the use of a do/while loop I set up a menu of choices for the user to make. Since the code will be based on the choice the user makes I needed to give an initial value to the variable *choice*. Type:

```
do
{
    try
    {
        System.out.println("\n\n\t\tChoose from the following menu\n\n\t\t1) Deposit
to Checking Account\n\n\t\t2) Withdrawal from Checking Account\n\n\t\t3) Deposit to
Savings Account\n\n\t\t4) Withdrawal from Savings Account\n\n\t\t5) Exit Program");
        choice = keyboard.nextInt();
```

- For the line for `System.out.println`, allow the software to automatically wrap. Do not hit the Enter key. Notice that I used the escape character of `\n` to move text to the next line and to add extra blank lines.

9. The cases in our switch statement will be based on the value of the variable *choice*. Type:

```
switch(choice)
{
```

```

case 1:
//request how much is to be deposited into the checking account and pass amount
//to the setBalance() method.

System.out.println("How much do you want to deposit to your checking account?");
checkingDeposit = keyboard.nextDouble();

checkingBalance = checkingBalance + checkingDeposit;

c1.setBalance(checkingBalance);

//set the value of the Date object which represents the date accessed
dateAccessed = new Date();

c1.setDateAccessed(dateAccessed);

System.out.println(c1);

break;

case 2:
//request amount to be withdrawn from checking account
System.out.println("How much do you wish to withdrawal from your checking account?");
checkingWithdrawal = keyboard.nextDouble();

//set the date accessed and pass to the setDateAccessed() method
dateAccessed = new Date();
c1.setDateAccessed(dateAccessed);

//check to see if the withdrawal is greater than the balance

if(checkingWithdrawal > c1.getBalance())
    checkingOverage = checkingWithdrawal - c1.getBalance();

//check to make certain the limit for overage is not exceeded
if(checkingOverage >= c1.getLimit())
{
    System.out.println("You are over you maximum overage limit. This withdrawal cannot be made. See
your bank manager.");

    break; //if over the limit break the case and display the menu
}
else
{
    checkingBalance = checkingBalance - checkingWithdrawal;
    c1.setBalance(checkingBalance);
}

System.out.println(c1);

break;

```

```

case 3:
//request amount to be deposited into savings account

    System.out.println("How much do you want to deposit to your savings account?");
    savingDeposit = keyboard.nextDouble();

    savingBalance = savingBalance + savingDeposit;

    dateAccessed = new Date();
    s1.setDateAccessed(dateAccessed);

    s1.setBalance(savingBalance);

    System.out.println(s1);

break;

case 4:
//request amount to be withdrawn from savings account. Savings cannot be
//overdrawn
    System.out.println("How much do you wish to withdrawal from your savings account?");
    savingWithdrawal = keyboard.nextDouble();

    dateAccessed = new Date();
    s1.setDateAccessed(dateAccessed);

    if(savingWithdrawal > s1.getBalance())
        System.out.println("Insufficient funds! Your savings balance is " + s1.getBalance() + " this is the most
you can withdrawal");
    else
    {
        savingBalance = savingBalance - savingWithdrawal;
        s1.setBalance(savingBalance);
    }

    System.out.println(s1);
break;

case 5:

//if user chose 5 from the menu the program exits
    System.exit(0);
break;

default:
//always include a default case should the user enter an incorrect choice
    System.out.println("Error! This is not a menu choice");
break;

} //close switch statement

```

```
//close try statement
```

- I have added a couple of comments throughout. However the code is easy to follow to see what is occurring.

10. If we have a try statement a catch statement must follow. Type:

```
catch(NumberFormatException e)
{
    System.out.println("Currency format must be used");
}
```

11. We close our loop, main method and class by typing:

```
    }while(choice !=5); //close do/while loop
} //close main method
} //close class
```

- Notice that the while portion of our do/while loop is based on whether the variable *choice* does not equal 5.

The Checking class

The Checking class will be in the same document. When this is the case the access modifier of public is omitted. A class that is public must be in its own file. This class is simple to understand so I will give you the entire code for it. Type:

```
class Checkings extends Account
{
    protected int overdraftLimit = 5000;

    protected DecimalFormat currency = new DecimalFormat("$##,##0.00");

    protected DecimalFormat percentage = new DecimalFormat("##,##0.0%");

    protected final double INTERESTRATE = 0.015;

    //default constructor method

    Checkings()
    {
    }
}
```

```

//overload the method

Checkings(int ID, double startingBalance)
{

    super(ID, startingBalance);

}

public int getLimit()
{

    return overdraftLimit;

}
//include a toString() method to display values of all attributes

public String toString()
{

    if(getDateAccessed() == null)
        setDateAccessed(getDateCreated());

    setAnnualInterestRate(INTERESTRATE);

    return "\n\n\t\tChecking Account\n\nID : " + getId() + "\nBalance: " + currency.format(getBalance()) + "\nDate created: "
+ getDateCreated() +

        "\nLast Date Accessed: " + getDateAccessed() + "\nAnnual Interest Rate: "
+ percentage.format(getAnnualInterestRate());

}

}

```

- The code within this class is simple to understand and is nothing you have not had before.

The Savings class is very similar to the Checking class. Type:

```

class Saving extends Account
{

    protected int overdraftLimit = 0;

    protected DecimalFormat currency = new DecimalFormat("$###,###0.00");

    protected DecimalFormat percentage = new DecimalFormat("##,###0.0%");

    protected final double INTERESTRATE = 0.045;

```



```

//default constructor method
Saving()
{

}

//overload the method

Saving(int ID, double startingBalance)
{
    super(ID, startingBalance);
}

public int getLimit()
{

    return overdraftLimit;
}

//create a toString() method to display the values of variables

public String toString()
{

    if(getDateAccessed() == null)
        setDateAccessed(getDateCreated());

    setAnnualInterestRate(INTERESTRATE);

    return "\n\n\tSavings Account\n\nID : " + getId() + "\nBalance: " + currency.format(getBalance()) + "\nDate created: " + getDateCreated() + "\nLast Date Accessed: " + getDateAccessed() + "\nAnnual Interest Rate: " + percentage.format(getAnnualInterestRate());
}
}

```

1. Compile the program, fix any errors if necessary, and run and test the program.
2. Compress ALL files into a single zip or rar file and submit it.