Programming Assignment 3-2

In this exercise, you will extend the functionality of the Employee class that was mentioned in the book and lecture slides. For this exercise, we imagine that the bank (a "credit union") provides services to employees of our (imaginary) company for three kinds of accounts: savings, checking, and retirement. We will therefore equip our Employee class with one instance field for each type of account. (In a real application, we would implement multiple accounts by storing them in some kind of List container – more on this later.) You will use the Account class that you developed in Lab 3-1 in conjunction with this version of Employee. Also, for practice, replace all uses of Date and GregorianCalendar with the new Date and Time API class LocalDate.

```
public class Employee {

   private Account savingsAcct;
   private Account checkingAcct;
   private Account retirementAcct;
   private String name;
   private LocalDate hireDate;

   public Employee(String name, int yearOfHire,
                       int monthOfHire, int dayOfHire){
      this.name = name;
      //Replace these lines with LocalDate references
      //GregorianCalendar cal =
      //new GregorianCalendar(yearOfHire,monthOfHire-1,dayOfHire);
      //hireDate = cal.getTime();
   }
   public void createNewChecking(double startAmount) {
      // implement

   }
   public void createNewSavings(double startAmount) {
      // implement

   }
   public void createNewRetirement(double startAmount) {
      // implement

   }
   public void deposit(AccountType acctType, double amt){
      // implement
   }
   public boolean withdraw(AccountType acctType, double amt){
      // implement
   }
   public String getFormattedAcctInfo() {
      // implement
      return null;
   }

}
```

The class file shows that employee now will provide new services. An `Employee` instance can be asked to create a new checking account, new savings account, or new retirement account. It can also be asked to make a deposit or make a withdrawal. In this exercise, you will implement these new methods.

The overall purpose of this new application is to make it possible for an authorized user to see a list of all employees along with each of their bank accounts. In the first phase of implementation, you will present output on the console. In the second phase (which will be the topic of a future lab), you will present it in a GUI.

To get the process started, a test class called `Main` (this is like the book's test class `EmployeeTest`) will create a number of instances of `Employee` (using imaginary names and other data) and will also populate each `Employee` instance with several accounts (that is, `Account` instances). It will then ask the user if he wants to see a formatted report of all accounts; if the user answers yes, the `Main` object will call the `getFormattedAcctInfo` on each `Employee` object and prepare a `String` to display in the console.

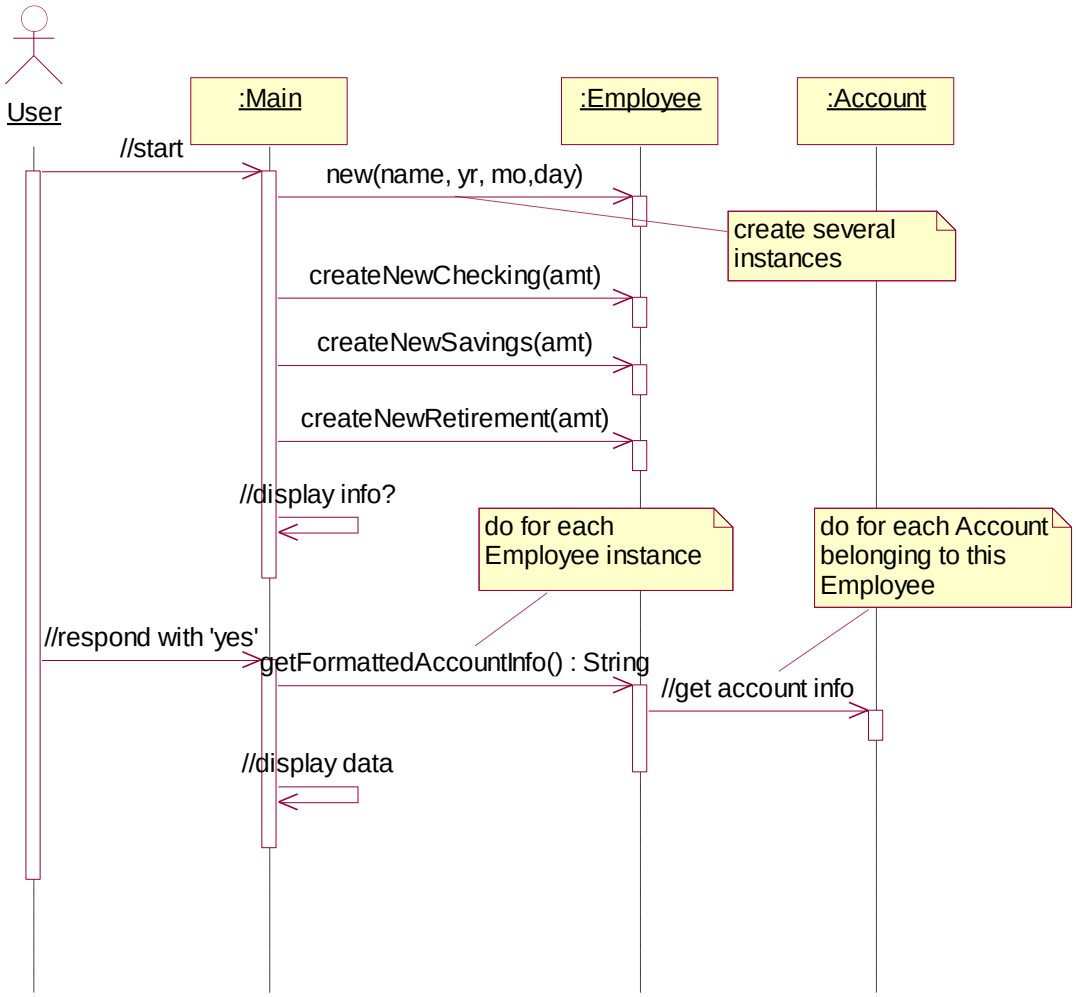
```
public class Main {
   Employee[] emps = null;
   public static void main(String[] args) {
      new Main();
   }
   Main(){
      emps = new Employee[3];
      emps[0] = new Employee("Jim Daley", 2000, 9, 4);
      emps[1] = new Employee("Bob Reuben", 1998, 1, 5);
      emps[2] = new Employee("Susan Randolph", 1997, 2,13);

      emps[0].createNewChecking(10500);
      emps[0].createNewSavings(1000);
      emps[0].createNewRetirement(9300);
      emps[1].createNewChecking(34000);
      emps[1].createNewSavings(27000);
      emps[2].createNewChecking(10038);
      emps[2].createNewSavings(12600);
      emps[2].createNewRetirement(9000);

      // for phase I - console output
      Scanner sc = new Scanner(System.in);
      System.out.print("See a report of all account balances? (y/n) ");
      String answer = sc.next();
      if(answer.equalsIgnoreCase("y")){
         String info = getFormattedAccountInfo();
         //display info to console
      }
      else {
         //do nothing..the application ends here
      }
   }
```

```
   String getFormattedAccountInfo() {
      //implement
      return null;
   }
}
```

The following diagram, known as a *sequence diagram*, illustrates the flow of execution:



What you need to do:

For Phase I, you will need to fill in code in the classes `Main` and `Employee` (plan on using the `Account` class that you created in Lab 3-1). You will notice that `Employee` has been placed in the `employeeinfo` package (place your `Account` class in this package too), but `Main` lies outside of this package. Do not modify this structural arrangement.

Main

If the user answers 'yes', you need to call the `getFormattedAccountInfo` method in `Main`, and then display the return value to the console. You will also need to implement the method `getFormattedAccountInfo`. This method will call the `getFormattedAcctInfo` method of each of the `Employee` instances, and then put those `String`s together into a good format.

Employee

- implement each of the `createNewXXX` methods by creating a new instance of `Account` with the appropriate data, and storing the new instance in the appropriate instance variable in `Employee`

- implement the `deposit()` method by calling the `makeDeposit()` method on the appropriate `Account` instance.

- implement the `withdraw()` method as follows: Call the appropriate `makeWithdrawal` method on the appropriate `Account` instance and then use the return value as the new return value for `withdraw()`.

- implement the `getFormattedAccountInfo()` by calling the `toString()` method on each `Account` instance to provide its own formatted representation of its own account type and balance. The `getFormattedAccountInfo` method should then piece these 3 `String`s together, and return the result. Note: your `Account` class already has an implementation of `toString();` you may need to modify the implementation of this method in order to obtain the desired output format.

- add "getter" methods for the `name` and `hireDate` fields.

Here is the expected output of your program, in the console application (for Phase I):

```
See a report of all account balances? (y/n) y
ACCOUNT INFO FOR Jim Daley:

Account type: checking
Current bal:  10500.0
Account type: savings
Current bal:  1000.0
Account type: retirement
Current bal:  9300.0

ACCOUNT INFO FOR Bob Reuben:

Account type: checking
Current bal:  34000.0
Account type: savings
Current bal:  27000.0

ACCOUNT INFO FOR Susan Randolph:
```

```
Account type: checking
Current bal:  10038.0
Account type: savings
Current bal:  12600.0
Account type: retirement
Current bal:  9000.0
```

Class Diagram. The following diagram – called a *class diagram* -- displays the instance fields and instance methods of classes `Main`, `Employee`, and `Account` and shows the simple relationships between these classes. The arrows indicate directions of "messages" from one class to another; if a line has no arrow, it means that "messages" can be passed in either direction. Notice that `Main` communicates only with `Employee`, not with `Account`.