Programming Assignment 3-1

Below, a class `Account` is defined, and the class `Employee`, as defined in the lecture, is shown.

```
class Account {
   public final static String CHECKING = "checking";
   public final static String SAVINGS = "savings";
   public final static String RETIREMENT = "retirement";
   private final static double DEFAULT_BALANCE = 0.0;
   private double balance;
   private String acctType;
   private Employee employee;

   Account(Employee emp, String acctType, double balance){
      employee = emp;
      this.acctType =acctType;
      this.balance = balance;
   }
   Account(Employee emp, String acctType){
      this(emp,acctType,DEFAULT_BALANCE);


   }
   public String toString() {
      return "type = " + acctType + ", balance = " + balance;
   }

   public void makeDeposit(double deposit) {
      //implement
   }
   public boolean makeWithdrawal(double amount) {
      //implement
   }
}
//same as the Employee class defined in the lecture
class Employee {
   //constructor
   Employee(String aName,
            double aSalary,
            int aYear,
            int aMonth,
            int aDay) {
      name = aName;
      salary = aSalary;
      GregorianCalendar cal =
            new GregorianCalendar(aYear,aMonth-1,aDay);
      hireDay = cal.getTime();
   }

   // instance methods
   public String getName() {
      return name;
   }
   public double getSalary() {
      return salary;
   }
   //needs to be improved!!
   public Date getHireDay() {
      return hireDay;
   }
   public void raiseSalary(double byPercent) {
      double raise = salary * byPercent / 100;
      salary += raise;
   }

   //instance fields
   private String name;
   private double salary;
   private Date hireDay;
}
```

The `Employee` class discussed in the lecture is reproduced here. To create an instance of `Account`, you must pass in an `Employee` and a `String` name for an account type, and, optionally, the starting balance. If you do not specify a starting balance, then the default balance (which is defined by a constant, and initially set to the value 0.0) is used. The possible kinds of account type are indicated by three public static constants – CHECKING, SAVINGS, and RETIREMENT. These constants should always be used whenever an account type needs to be specified by name.

In this assignment, do the following:

1. Refactor the Account class so that the three account types CHECKING, SAVINGS, RETIREMENT, are the three instances of an enumerated type called `AccountType;` like Java classes, the `enum` should be placed in a separate file. After defining this `enum` and removing the account types from the `Account` class, make the necessary changes to instance variables and the constructors of `Account`.

2. Add the following methods to the `Account` class:
   ```
   //updates the balance field
   public void makeDeposit(double val);

   //updates the balance field and returns true, unless
   //withdrawal amount is too large; in that case,
   //it does not modify the balance field, and returns false
   public boolean makeWithdrawal(double amount)
   ```

3. Add public accessor methods for the fields `acctType` and `balance`.

4. Correct the implementation of `getHireDay()` in `Employee`, as discussed in the lecture.

5. Create a class `Main` having a `main` method that does the following:

   a. It creates a new `Employee` object `employee` (you can invent your own name, hireday, salary, etc., to be used in the constructor)

b. Then it creates a checking account, savings account and retirement account for `employee`, each with a starting balance of $300.
c. Then it prints to the console the account data for each of these accounts (making use of the `toString()` method that has been provided in `Account`)