Chapter 5: Enhancing Classes
Lab Exercises

A Flexible Account Class

In this lab, you will be modifying a class which represents a bank account. You will be using this class for several labs.

Create a folder `AccountProject` in your `\Chapter 5` folder. In the `AccountProject` folder, create a new class, `Account.java` and copy the code below into the `Account` class.

- `Account.java` is the class which defines methods to withdraw, deposit, get the balance and account number, and return a String representation of the Account object.

1. Review the code for the two classes. Compile and run the program to create one account.

2. Overload the constructor as follows:

   - `public Account (double initBal, String owner)` – initializes the balance and owner as specified; randomly generates the account number between 1000 and 9999 (use the `Math.random()` method).
   - `public Account (String owner)` – initializes the owner as specified; sets the initial balance to 0 and randomly generates the account number between 1000 and 9999.
   - Compile your code and correct any errors.

3. Overload the *withdraw* method with one that will accept two doubles: the amount to be withdrawn and a fee. Both amounts are to be deducted from the account. Add a `println` statement to the withdraw methods to notify the user a withdraw from the account is being performed. Use the following as an example:

```
Withdraw $500 from John’s account.  New balance is $600.
Withdraw $500 plus $2 fee from John’s account.  New balance is $598.
```

4. Add a `toString`() method to `Account.java` that returns a formatted string containing the name, account number and balance of the account. For example:

```
John’s account number is 1234. Balance is $600.
```

5. Create a tester program, `AccountTest.java`. Create three new `Account` objects `acct1`, `acct2` and `acct3`, using the information in the table below, invoking each of the three different constructor methods. Perform the tasks on each account as listed below.

acct1	acct2	acct3
create with $1000, “John”, 1234	create with “Sally”	create with “Tommy”, $5500
deposit $100	deposit $5000	withdraw $500
withdraw $500	withdraw $150 plus $2.50 fee	print object
print object	try to withdraw $5000	
	print object	

```
//*******************************************************
// Account.java
//
// A bank account class with methods to deposit to, withdraw from,
// change the name on, and get a String representation
// of the account.
//*******************************************************

public class Account
{
  private double balance;
  private String name;
  private int acctNum;

  //----------------------------------------------
  //Constructor -- initializes balance, owner, and account number
  //----------------------------------------------
  public Account(double initBal, String owner, int number)
  {
    balance = initBal;
    name = owner;
    acctNum = number;
  }

  //----------------------------------------------
  // Checks to see if balance is sufficient for withdrawal.
  // If so, decrements balance by amount; if not, prints message.
  //----------------------------------------------
  public void withdraw(double amount)
  {
    if (balance >= amount)
       balance -= amount;
    else
       System.out.println("Insufficient funds");
  }

  //----------------------------------------------
  // Adds deposit amount to balance.
  //----------------------------------------------
  public void deposit(double amount)
  {
    balance += amount;
  }

  //----------------------------------------------
  // Returns balance.
  //----------------------------------------------
  public double getBalance()
  {
    return balance;
  }

 //----------------------------------------------
  // Returns account name.
  //----------------------------------------------
  public String getName()
  {
    return name;
  }
}
```

Opening Accounts

File `Account.java` (see previous exercise) contains a definition for a simple bank account class with methods to withdraw, deposit, get the balance and account number, and return a String representation.

Make a new project folder `\AccountProject2` and copy the `Account.class` file into the folder. Modify the `Account.class` program to do the following:

1. What will happen if the randomly generated account number already exists? Use a static variable to hold the next account number instead.

   a. Declare a `private static` integer variable `nextAccountNum` to hold this value. Initialize to 1000 when it is declared.
   b. Modify the two constructors which generate a random number to assign the account number from this variable and to increment this variable every time an account is created. Still allow accounts to be created by passing an account number.
   c. Add a `static` method `getNextNum` that returns the next id number. Think about why this method should be static – its information is not related to any particular account.

2. Write a test program, `AcountTest2`, that prompts for the number of accounts to be created.
3. Loop and prompt the user for a name. Create an account with an initial balance of $100.
4. Print each account object after it is created and the next id number to be assigned (use the `static` method in the `Account` class to return the number of accounts, do not invoke the static method through the object). Note: You will use the same Account reference variable for each object created. Account objects will be created, printed and then deleted with the next Account. Use the following output as a guide (user input is in red):

```
How many accounts will you be creating? 3

Enter the name of account #1: John

John’s account number is 1000. Balance is $100.
Next id number is: 1001

Enter the name of account #2: Mike

Mikes’s account number is 1001. Balance is $100.
Next id number is: 1002

Enter the name of account #3: Kathy

Kathy’s account number is 1002. Balance is $100.
Next id number is: 1003

End of program.  Goodbye.
```

Transfering Funds

File `Account.java` (see previous exercise) contains a definition for a simple bank account class with methods to withdraw, deposit, get the balance and account number, and return a String representation.

Make a new project folder `\TransferProject` and copy the `Account.java` file. Modify the `Account.java` program to do the following:

1. Add a method `public void transfer(Account acct, double amount)` to the `Account` class that allows the user to transfer funds from one bank account to another. If `acct1` and `acct2` are `Account` objects, then the call `acct1.transfer(acct2,957.80)` should transfer $957.80 from `acct1` to `acct2`. Be sure to clearly document which way the transfer goes! Add a println statement to the method to print out the transfer details:

   ```
   Transferred $957.80 from John’s account to Sally’s account.
   ```

2. Write a class `TransferTest` with a main method that creates two bank account objects (John and Sally) and enters a loop that does the following:
   - Asks if the user would like to transfer from account1 to account2, transfer from account2 to account1, or quit. Use the `getName()` method in the account object and use the example dialog as a guide:

     ```
     1 – Transfer from John to Sally’s account.
     2 – Transfer from Sally to John’s account.
     3 - Quit
     Enter the number to choose an option from above:
     ```

   - If a transfer is chosen, asks the amount of the transfer, carries out the operation, and prints the new balance for each account.
   - Repeats until the user asks to quit, then prints a summary for each account.

3. Add a static method to the `Account` class that lets the user transfer money between two accounts without going through either account. You can (and should) call the method `transfer` just like the other one – you are overloading this method, except you will invoke the method through the `Account` class name, NOT THE OBJECT. Your new method should take two `Account` objects and an amount and transfer the amount from the first account to the second account. The signature will look like this:

   ```
   public static void transfer(Account acct1, Account acct2, double amount)
   ```

   Modify your `TransferTest` class to test your new code by adding another loop (see step 2 above), using the `static transfer` method. Print a statement to denote the use of the static method:

   ```
   ****** Testing static transfer method ******
   ```

4. Document the code in your tester program. Your output should be clear and easy to read.

Setting Priorities

Folder: `\PriorityProject`   Files you create: `Priority.java, Task.java, TaskTest.java`

You are going to be designing a program to assign priorities to a varity of tasks. Each task will be it's own object and will be created from a class `Task`. Priorities can range from 1-10, 1 being the lowest priority (least important task) and 10 being the highest priority (most important task).

Prioritize the tasks listed below. Each task can be assigned a priority from 1-10 and tasks can have the same priorty value.

Eat
Sleep
Finish APCS programs
Exercise
Watch TV

1. Design a Java interface called `Priority` that has two methods: `setPriority` and `getPriority`. Priority should have three static consants, MIN_PRIORITY, MED_PRIORITY and MAX_PRIORITY, set to 1, 5 and 10 respectivly.

2. Design and implement a class called `Task` that implements the `Priority` interface. The `Task` class should have the following instance variables and methods:

   - A variable to hold the priority.
   - A variable to hold the name of the task.
   - A contructor that accepts the name of the task as a parameter.
   - A method to return the name of the task.
   - A method to set the priority of the task.
   - A method to return the priority of the task.
   - A `toString` method which returns a string with the task name and priority.

3. Create a driver class `TaskTest` to create your task objects and set their priorities. Set priorities by passing a value from 1-10 and by passing one of the static constants defined in the interface. For example:

```
t1.setPriority(3);                         // sets priority to 3
t2.setPriority(Priority.MIN_PRIORITY);     // sets priority to 1
```

4. Print out each task and it's priority.

5. Modify the `Task` class to implement the `Comparable` interface. Provide the code for the `compareTo()` method to compare the priority variables.

6. Write a helper static method, `comparePriorities(),` in your `TaskTest` class which is passed two `Task` objects, and prints the appropriate message based on the results of calling `compareTo()`. Your `main` method will call the helper method:

```
comparePriorities(t1,t2);
comparePriorities(t1,t3);
comparePriorities(t1,t4);
```

7. Your output should look something like the following:

```
"Finish APCS programs" has a higher priority than "Eat"
"Finish APCS programs" has a lower priority than "Sleep"
"Finish APCS programs" has an equal priority to "Exercise"
```

   Your output will look different, depending on the priorites you set for each task.

Finch Visits the Farm

The Finch has visited a farm recently and would like to mimic the animals at the farm. Could you help the Finch do this? Make a driver program, `FarmFinch` that has the following *static* methods:

```
public static boolean playAnimal(String animal)


 public static void Cow()

 public static void Horse()

 public static void Duck()

 public static void Sheep()
```

Create the Finch object outside of the main method as a static reference variable:

```
      public static Finch myFinch = new Finch();
```

Method `playAnimal` is boolean - it should return `true` if a valid animal has been passed (Cow, Horse, Duck, or Sheep), and `false` otherwise.

Methods for the different animals should do their best to have the Finch imitate the animal through movement, sound, and light. You may find this site useful in conjunction with the `playWav` method.

```
import edu.cmu.ri.createlab.terk.robot.finch.Finch;

public class FarmFinch
{
   public static Finch myFinch = new Finch();
   public static void main(final String[] args)
   {
      playAnimal("Duck");
        // Always end your program with finch.quit()
      myFinch.quit();
      System.exit(0);
   }

    public static boolean playAnimal(String a)
    {
        if (a.equals("Duck"))
        {
            duck();
            return true;
        }
        else return false;
    }

    public static void duck()
    {
        myFinch.saySomething("i am a duck");
            // Add code to make the finch behave like a duck.
    }
}
```