# Chapter 4: Writing Classes
# Lab Exercises

# Prelab Exercises

1. Constructors are special methods included in class definitions.
   a. What is a constructor used for?
   b. How do constructors differ from other methods in a class?

2. Both methods and variables in a class are declared as either *private* or *public*. Describe the difference between private and public and indicate how a programmer decides which parts of a class should be private and which public.

3. Consider a class that represents a bank account.
   a. Such a class might store information about the account balance, the name of the account holder, and an account number. What instance variables would you declare to hold this information? Give a type and name for each.
   b. There are a number of operations that would make sense for a bank account—withdraw money, deposit money, check the balance, and so on. Write a method header with return type, name, and parameter list, for each such operation described below. Don't write the whole method—just the header. They will all be public methods. The first one is done for you as an example.
      i. Withdraw a given amount from the account. This changes the account balance, but does not return a value.

         ```
         public void withdraw(double amount)
         ```

      ii. Deposit a given amount into the account. This changes the account balance, but does not return a value.
      iii. Get the balance from the account. This does not change anything in the account; it simply returns the balance.
      iv. Return a string containing the account information (name, account number, balance). This does not change anything in the account.
      v. Charge a $10 fee. This changes the account balance but does not return a value.
      vi. Create a new account given an initial balance, the name of the owner, and the account number. Note that this will be a constructor, and that a constructor does not have a return type.

# A Bank Account Class

1. File *Account.java* contains a partial definition for a class representing a bank account. Save it to your directory and study it to see what methods it contains. Then complete the Account class as described below. Note that you won't be able to test your methods until you write ManageAccounts in question #2.

   a. Fill in the code for method *toString*, which should return a string containing the name, account number, and balance for the account.
   b. Fill in the code for method *chargeFee*, which should deduct a service fee from the account.
   c. Modify *chargeFee* so that instead of returning void, it returns the new balance. Note that you will have to make changes in two places.
   d. Fill in the code for method *changeName* which takes a string as a parameter and changes the name on the account to be that string.

2. File *ManageAccounts.java* contains a shell program that uses the Account class above. Save it to your directory, and complete it as indicated by the comments.

3. Modify the Account class so it performs validity on the deposit and withdraw operations.
Specifically don't allow the deposit of negative number or withdrawal that exceeds the current balance.
Return appropriate error message if the these problem occurs.

Hint! In the methods deposit and withdraw, change so the method will return a Sting contain the appropriate error massage in place of using the System.out.print .

```
//********************************************************
// Account.java
//
// A bank account class with methods to deposit to, withdraw from,
// change the name on, charge a fee to, and print a summary of the account.
//********************************************************

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

  //-----------------------------------------
  //Constructor -- initializes balance, owner, and account number
  //-----------------------------------------
  public Account(double initBal, String owner, long 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 a string containing the name, account number, and balance.
  //-----------------------------------------------
  public String toString()
  {

  }

  //-----------------------------------------------
  // Deducts $10 service fee
  //-----------------------------------------------
  public void chargeFee()
  {


  }

  //-----------------------------------------------
  // Changes the name on the account
  //-----------------------------------------------
  public void changeName(String newName)

  {

  }

}
```

```
// ************************************************************
//   ManageAccounts.java
//
//   Use Account class to create and manage Sally and Joe's
//   bank accounts
// ************************************************************

public class ManageAccounts
{
    public static void main(String[] args)
    {
           Account acct1, acct2;

           //create account1 for Sally with $1000
           acct1 = new Account(1000, "Sally", 1111);

           //create account2 for Joe with $500

           //deposit $100 to Joe's account

           //print Joe's new balance (use getBalance())

           //withdraw $50 from Sally's account

           //print Sally's new balance (use getBalance())

           //charge fees to both accounts

           //change the name on Joe's account to Joseph

           //print summary for both accounts

    }
}
```

# Tracking Grades

A teacher wants a program to keep track of grades for students and decides to create a student class for his program as follows:

- Each student will be described by three pieces of data: his/her name, his/her score on test #1, and his/her score on test #2.
- There will be one constructor, which will have one argument—the name of the student.
- There will be three methods: *getName*, which will return the student's name; *inputGrades*, which will prompt for and read in the student's test grades; and *getAverage*, which will compute and return the student's average.

1. File *Student.java* contains an incomplete definition for the Student class. Save it to your directory and complete the class definition as follows:
   a. Declare the instance data (name, score for test1, and score for test2).
   b. Create a Scanner object for reading in the scores.
   c. Add the missing method headers.
   d. Add the missing method bodies.

2. File *Grades.java* contains a shell program that declares two Student objects. Save it to your directory and use the *inputGrades* method to read in each student's test scores, then use the *getAverage* method to find their average. Print the average with the student's name, e.g., "The average for Joe is 87." You can use the *getName* method to print the student's name.

3. Add statements to your Grades program that print the values of your Student variables directly, e.g.:

```
System.out.println("Student 1: " + student1);
```

This should compile, but notice what it does when you run it—nothing very useful! When an object is printed, Java looks for a *toString* method for that object. This method must have no parameters and must return a String. If such a method has been defined for this object, it is called and the string it returns is printed. Otherwise the default *toString* method, which is inherited from the Object class, is called; it simply returns a unique hexadecimal identifier for the object such as the ones you saw above.

Add a toString method to your Student class that returns a string containing the student's name and test scores, e.g.:

```
Name: Joe  Test1: 85  Test2: 91
```

Note that the toString method does not call System.out.println—it just returns a string.

Recompile your Student class and the Grades program (you shouldn't have to change the Grades program—you don't have to call toString explicitly). Now see what happens when you print a student object—much nicer!

```
// ****************************************************************
//   Student.java
//
//   Define a student class that stores name, score on test 1, and
//   score on test 2.  Methods prompt for and read in grades,
//   compute the average, and return a string containing student's info.
// ****************************************************************
import java.util.Scanner;


public class Student
{
    //declare instance data

    //-----------------------------------------------
    //constructor
    //-----------------------------------------------
    public Student(String studentName)
    {
           //add body of constructor
    }

    //-----------------------------------------------
    //inputGrades: prompt for and read in student's grades for test1 and
// test2.
    //Use name in prompts, e.g., "Enter's Joe's score for test1".
    //-----------------------------------------------

public void inputGrades()
    {
           //add body of inputGrades
    }

    //-----------------------------------------------
    //getAverage: compute and return the student's test average
    //-----------------------------------------------

    //add header for getAverage
    {
           //add body of getAverage
    }

    //-----------------------------------------------
    //getName: print the student's name
    //-----------------------------------------------

    //add header for printName
    {
           //add body of printName
    }

}
```

```
// ************************************************************
//   Grades.java
//
//   Use Student class to get test grades for two students
//   and compute averages
//
// ************************************************************
public class Grades
{
    public static void main(String[] args)
    {
           Student student1 = new Student("Mary");
           //create student2, "Mike"

           //input grades for Mary
           //print average for Mary

           System.out.println();

           //input grades for Mike
           //print average for Mike

    }
}
```

# Bank Dispenser

Go back to your Account class. Study and remember the methods.

1. Add to your class Account a new method *public long getAcctNum ()*{ // body }. The method should return the acctNum.

2. Now you will use your Account class for really doing bank activities.
In my programme below, I have done a BankDispenser program. The bank contain only one Account-object and a menu to operate with the accounts.

Copy, save, compile and run the programme below. Pay attention to my implementation and try to understand how the programme works. Case 1 and 3 are implemented.
In the same way, implement case 2. Run your programme.

```
import java.util.Scanner;

public class BankDispenser
{
      static Scanner scan = new Scanner(System.in);
      static Account account1=new Account( 1000, "Svensson Dan", 750123);

public static void main(String[] args)
    {
      printMenu();
      int choice = scan.nextInt();
```

```
    while (choice != 0)
      {
      dispatch(choice);
      printMenu();
      choice = scan.nextInt();
       }
}

public static void dispatch(int choice)
    {
     switch(choice)
       {
      case 0:
      System.out.println("Bye!");
      break;
      case 1:{
      System.out.println("Insert your account number");
      int nr = scan.nextInt();

// the program checks if the account number is correct. If yes the
//programme asks for the deposit amount and operate it.
      if( account1.getAcctNum()==nr)
      {
      System.out.println("Insert the amount to deposit");
      int amount=scan.nextInt();
      account1.deposit(amount);
      }
      else
      System.out.println( " Wrong account number");
      break;}
      case 2:
       // your code for withdraw
      break;


       case 3: {
       System.out.println("Insert your account number");
       int nr = scan.nextInt();
       if( account1.getAcctNum()==nr)
       System.out.println( account1.toString());
       break;}
       default:
      System.out.println("Sorry, invalid choice");
       }
    }
    //----------------------------
    // Print the user's choices
    //----------------------------
    public static void printMenu()
    {
   System.out.println("\n  Welcome to MY Bank ");
   System.out.println("   ====");
   System.out.println("0: Quit");
   System.out.println("1: Deposit" );
   System.out.println("2: Withdraw");
   System.out.println("3: Show account information");

   System.out.print("\nEnter your choice: ");
    }
}
```

3. As you could see I only have one Account object in the programme. Now create one more Accout-object and added to your BankDispenser, for example:

```
static Account account2=new Account( 2000, "Anderson Amilia",
881203);
```

Now, change in the BankDispenser so the program can to operations in both accounts.
Be sure you always check the account number before the programme allows any bank operations.

# Drawing Squares ( Ej Obligatoriskt)

1. Write a class Square that represents a square to be drawn. Store the following information in instance variables:
   - size (length of any side)
   - x coord of upper left-hand corner
   - y coord of upper left-hand corner
   - color

```
//*********************************************************
//Square.java
//
// A Square-object rpresents a square to be drawn.
//
//*********************************************************

import java.awt.*;

public class Square
{
  private int size;
  private int x;
  private int y;
  private Color color

  //-----------------------------------------------
  //A parameterless constructor that generates random values for the size,
color,
  //x, and y. Make the size between 50 and 100, x between 0 and 300, y
between 0
  //and 300. The squares can be any color—note that you can pass a single
int
  //parameter to the Color constructor, but it will only consider the first
24 bits
  //(8 bits for each of R, G, B component), as follow.
  //   color =new  Color( 33, 100,76)

  //-----------------------------------------------
  public Square()
  {
     //your code here
```

```
  }

  //-----------------------------------------------
   //A draw method that draws the square at its x,y coordinate in its
color. Note
   // that you need a Graphics- object (pen)  as you see in the method
   //specification, than use pen.fillRect(…)metod to draw.
 //-----------------------------------------------
  public void draw(Graphics pen)
  {
     //your code here
  }


  }
```

- IMPORTANT: Your random number generator must be declared at the class level (not inside the constructor), and must be declared *static*. So its declaration and initialization should appear with the declarations of size, color, x, and y, and should look like this:

```
private static Random generator = new Random();
```

2. **Now write an applet DrawSquares** that uses your Square class to create and draw 5 squares. This code should be very simple; the *paint* method will simply create 5 Squares and then draw it. Don't forget to pass the Graphics object to *draw*.

# Voting with Buttons ( Ej Obligatoriskt)

Files *VoteCounter.java* and *VoteCounterPanel.java* contain slightly modified versions of *PushCounter.java* and *PushCounterPanel.java* in listings 4.10 and 4.11 of the text. As in the text the program counts the number of times the button is pushed; however, it assumes ("pretends") each push is a vote for Joe so the button and variables have been renamed appropriately.

1. Compile the program, then run it to see how it works.

2. Modify the program so that there are two candidates to vote for—Joe and Sam. To do this you need to do the following:
   a. Add variables for Sam—a vote counter, a button, and a label.
   b. Add a new inner class named *SamButtonListener* to listen for clicks on the button for Sam. Instantiate an instance of the class when adding the ActionListener to the button for Sam.
   c. Add the button and label for Sam to the panel.

3. Compile and run the program.

```
//********************************************************
// VoteCounter.java
//
// Demonstrates a graphical user interface and event listeners to
// tally votes for two candidates, Joe and Sam.
//********************************************************
import javax.swing.JFrame;

public class VoteCounter
{
    //-----------------------------------------------
    // Creates the main program frame.
    //-----------------------------------------------
    public static void main(String[] args)
    {
        JFrame frame = new JFrame("Vote Counter");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        frame.getContentPane().add(new VoteCounterPanel());

        frame.pack();
        frame.setVisible(true);
    }
}
```

```
//********************************************************
// VoteCounterPanel.java
//
// Demonstrates a graphical user interface and event listeners to
// tally votes for two candidates, Joe and Sam.
//********************************************************

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class VoteCounterPanel extends JPanel
{
    private int votesForJoe;
    private JButton joe;
    private JLabel labelJoe;

    //-----------------------------------------------
    // Constructor: Sets up the GUI.
    //-----------------------------------------------
    public VoteCounterPanel()
    {
        votesForJoe = 0;

        joe = new JButton("Vote for Joe");
        joe.addActionListener(new JoeButtonListener());

        labelJoe = new JLabel("Votes for Joe: " + votesForJoe);

        add(joe);
        add(labelJoe);

        setPreferredSize(new Dimension(300, 40));
        setBackground(Color.cyan);
    }

    //*************************************************
    // Represents a listener for button push (action) events
    //*************************************************
    private class JoeButtonListener implements ActionListener
    {
        //-----------------------------------------
        // Updates the counter and label when Vote for Joe
        // button is pushed
        //-----------------------------------------
        public void actionPerformed(ActionEvent event)
        {
            votesForJoe++;
            labelJoe.setText("Votes for Joe: " + votesForJoe);
        }
    }
}
```