Chapter 5: Conditionals and Loops
Lab Exercises

Activities at Lake LazyDays

As activity directory at Lake LazyDays Resort, it is your job to suggest appropriate activities to guests based on the weather:

```
temp >= 80:        swimming
60 <= temp < 80:   tennis
40 <= temp < 60:   golf
temp < 40:         skiing
```

1. Write a program that prompts the user for a temperature, then prints out the activity appropriate for that temperature. Use a cascading if, and be sure that your conditions are no more complex than necessary.

Rock, Paper, Scissors

Program *Rock.java* contains a skeleton for the game Rock, Paper, Scissors. Open it and save it to your directory. Add statements to the program as indicated by the comments so that the program asks the user to enter a play, generates a random play for the computer, compares them and announces the winner (and why). For example, one run of your program might look like this:

```
$ java Rock
Enter your play: R, P, or S
r
Computer play is S
Rock crushes scissors, you win!
```

Note that the user should be able to enter either upper or lower case r, p, and s. The user's play is stored as a string to make it easy to convert whatever is entered to upper case. Use a switch statement to convert the randomly generated integer for the computer's play to a string.

```
// ****************************************************************
//   Rock.java
//
//   Play Rock, Paper, Scissors with the user
//
// ****************************************************************
import java.util.Scanner;
import java.util.Random;

public class Rock
{
    public static void main(String[] args)
    {
           String personPlay;    //User's play -- "R", "P", or "S"
           String computerPlay;  //Computer's play -- "R", "P", or "S"
           int computerInt;      //Randomly generated number used to
determine
                                 //computer's play

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

```
            Random generator = new Random();

            //Get player's play -- note that this is stored as a string
            //Make player's play uppercase for ease of comparison
            //Generate computer's play (0,1,2)
            //Translate computer's randomly generated play to string
            switch (computerInt)
      {

            }

            //Print computer's play
            //See who won.  Use nested ifs instead of &&.
            if (personPlay.equals(computerPlay))
                System.out.println("It's a tie!");
            else if (personPlay.equals("R"))
                if (computerPlay.equals("S"))
                       System.out.println("Rock crushes scissors.  You
win!!");
                else

                       //...  Fill in rest of code
    }}


    }}
```

Password checker.

Write a program that reads in a string from the command line and checks whether it is a "good" password. Here, assume "good" means that it (i) is at least 8 characters long, (ii) contains at least one digit 0-9, (iii) contains at least one upper case letter, (iv) contains at least one lower case letter, and (v) contains at least one non-alphanumeric character.

The class Encrypt

In the lecture you saw how to implement a crypo algorithm by using ascii codes for the characters.

```
import java.util.*;
    public class Encrypt
   {

       public static void main (String [] arg)
      {

          Scanner scan= new Scanner( System.in);
          System.out.println (" Write a text to enkrypt");
          String text =scan.nextLine();
          String crypt_text="";
```

```
            for(int i=0;i<text.length();i++){

                int ascii_ny=(int)text.charAt(i)+1;

                krypt_text=krypt_text+((char)ascii_ny);
            }
            System.out.println(krypt_text);
        }
    }
```

1. Modify the program as follow: Write a method
public static String encrypt(String text) { …. }. The method takes as input a text String, encrypt the text as you see in my programme and then return the encrypted text. Call the method in your main. I hope the programme works as before.

2. Write a new method **public static String decrypt(String text) { …. }.** The method takes as input an encrypted text String, decrypt the text and return the decrypted text
The code to decrypt is almost the same as encrypt, isn't it ?
Call this method in your programme and show that you get the same text back after decryption.

A class UsefulMethods

In a class called UsefulMethods :

a) Write a method public static int maxOfTwo(int tal1, int tal2) {} . As you can see this method will accept two integers as parameters and returns the larger of this two.
b) In another class called TestUsefulMethods, write a main program and test your maxOfTwo (). To do it , first read two values from the user (Sanner object required) and then call maxOfTwo method with the two values. The programme shoul print the return from this method.

c) Go back to UsefulMethods class and write another method. This method will also be static, and the name should be evenlyDivisible. The method will accept two integers as parameters and return true if the first parameter is evenly divisible by the second or vice versa and false if not.

d) Test your method by calling it in your program TestUsefulMethods. Print the result from the method.

e) Write a method public static int maxOfThre(int tal1, int tal2, int tal3) {} . As you can see this method will accept two integers as parameters and returns the larger of this two. Reuse the method maxOfTwo to implement this method.

f) Call this method in a main () and show that it works.

Funny Game

(A better guess game)
a) How many guesses does it take to guess a secret number between 1-N? For example I am thinking of a number between 1-100. I'll tell you whether your guess is too high or too low and so on.
Write a programme that let the user input a positive integer N. The programme will than generate a secret number between 1-N. The user will than try to guess the secret number. The help information from the program will be " TO HIGH" or "TO LOW" .
The programme will then report how many guesses it take to guess the number.
Use the Random class to generate numbers. Use Scanner class to read data from the user.

Multiplication

Your little sister / brother asks you to help her/him with the multiplication, and you decide to write a java application that tests her skills. The programme will let her input a starting number, such as 5. It will generate multiplication problems ranging from 5 X1 to 5X12. For each problem she will be prompted to enter the correct answer. The program should check the answer and should not let her/him advance to the next question until the correct answer is given.
To quit exercising this number and go further she/he should enter -1 .

You decide if you will have a user interface (frame or applet) or only textual interface (Scanner input and system output) to you programme.

Date Validation

In this exercise you will write a program that checks to see if a date entered by the user is a valid date in the second millenium. A skeleton of the program is in *Dates.java*. Open this program and save it to your directory. As indicated by the comments in the program, fill in the following:

1. An assignment statement that sets monthValid to true if the month entered is between 1 and 12, inclusive.
2. An assignment statement that sets yearValid to true if the year is between 1000 and 1999, inclusive.
3. An assignment statement that sets leapYear to true if the year is a leap year. Here is the leap year rule (there's more to it than you may have thought!):

   > If the year is divisible by 4, it's a leap year UNLESS it's divisible by 100, in which case it's not a leap year UNLESS it's divisible by 400, in which case it is a leap year. If the year is not divisible by 4, it's not a leap year.

   Put another way, it's a leap year if a) it's divisible by 400, or b) it's divisible by 4 and it's *not* divisible by 100. So 1600 and 1512 are leap years, but 1700 and 1514 are not.

4. An if statement that determines the number of days in the month entered and stores that value in variable daysInMonth. If the month entered is not valid, daysInMonth should get 0. Note that to figure out the number of days in February you'll need to check if it's a leap year.
5. An assignment statement that sets dayValid to true if the day entered is legal for the given month and year.
6. If the month, day, and year entered are all valid, print "Date is valid" and indicate whether or not it is a leap year. If any of the items entered is not valid, just print "Date is not valid" without any comment on leap year.

```
// ****************************************************************
// Dates.java
//
// Determine whether a 2nd-millenium date entered by the user
// is valid
// ****************************************************************
import java.util.Scanner;

public class Dates
{
    public static void main(String[] args)
    {
           int month, day, year;   //date read in from user
           int daysInMonth;        //number of days in month read in
           boolean monthValid, yearValid, dayValid;  //true if input from
user is valid
           boolean leapYear;       //true if user's year is a leap year

      Scanner scan = new Scanner(System.in);

           //Get integer month, day, and year from user

           //Check to see if month is valid

           //Check to see if year is valid

           //Determine whether it's a leap year

           //Determine number of days in month

           //User number of days in month to check to see if day is valid

           //Determine whether date is valid and print appropriate message

    }}
```

In a while loop, execution of a set of statements (the *body* of the loop) continues until the boolean expression controlling the loop (the *condition*) becomes false. As for an if statement, the condition must be enclosed in parentheses. For example, the loop below prints the numbers from 1 to to LIMIT:

```
final int LIMIT = 100;          // setup
int count = 1;

while (count <= LIMIT)          // condition
{                               // body
  System.out.println(count);    //   -- perform task
  count = count + 1;            //   -- update condition
}
```

Complemented DNA

Complemented DNA string. Write a program to read in a DNA string (A, C, T, G) and print out its complement (substitute A for T, T for A, C for G, and G for C). Hint: use replace several times, but be careful.

Counting Characters , testing switch statement (Ej Obligatoriskt)

The file *Count.java* contains the skeleton of a program to read in a string (a sentence or phrase) and count the number of blank spaces in the string. The program currently has the declarations and initializations and prints the results. All it needs is a loop to go through the string character by character and count (update the *countBlank* variable) the characters that are the blank space. Since we know how many characters there are (the *length* of the string) we use a count controlled loop—*for* loops are especially well-suited for this.

1. Add the *for* loop to the program. Inside the for loop you need to access each individual character—the *charAt* method of the String class lets you do that. The assignment statement

   ```
   ch = phrase.charAt(i);
   ```

   assigns the variable *ch* (type char) the character that is in index i of the String *phrase*. In your for loop you can use an assignment similar to this (replace i with your loop control variable if you use something other than i). NOTE: You could also directly use phrase.charAt(i) in your if (without assigning it to a variable).

2. Test your program on several phrases to make sure it is correct.

3. Now modify the program so that it will count several different characters, not just blank spaces. To keep things relatively simple we'll count the a's, e's, s's, and t's (both upper and lower case) in the string. You need to declare and initialize four additional counting variables (e.g. *countA* and so on). Your current *if* could be modified to cascade but another solution is to use a *switch* statement. Replace the current *if* with a switch that accounts for the 9 cases we want to count (upper and lower case a, e, s, t, and blank spaces). The cases will be based on the value of the *ch* variable. The switch starts as follows—complete it.

```
switch (ch)
{
   case 'a':
   case 'A':  countA++;
              break;

   case ....

}
```

Note that this switch uses the "fall through" feature of switch statements. If *ch* is an 'a' the first case matches and the switch continues execution until it encounters the *break* hence the countA variable would be incremented.

4. Add statements to print out all of the counts.

5. It would be nice to have the program let the user keep entering phrases rather than having to restart it every time. To do this we need another loop surrounding the current code. That is, the current loop will be nested inside the new loop. Add an outer while loop that will continue to execute as long as the user does NOT enter the phrase *quit*. Modify the prompt to tell the user to enter a phrase or *quit* to quit. Note that all of the initializations for the counts should be inside the while loop (that is we want the counts to start over for each new phrase entered by the user). All you need to do is add the while statement (and think about placement of your reads so the loop works correctly). Be sure to go through the program and properly indent after adding code—with nested loops the inner loop should be indented.

```
// ********************************************************
//   Count.java
//
//   This program reads in strings (phrases) and counts the
//   number of blank characters and certain other letters
//   in the phrase.
// ********************************************************

import java.util.Scanner;

public class Count
{
  public static void main (String[] args)
  {
      String phrase;    // a string of characters
      int countBlank;   // the number of blanks (spaces) in the phrase
      int length;       // the length of the phrase
      char ch;          // an individual character in the string

           Scanner scan = new Scanner(System.in);

      // Print a program header
      System.out.println ();
      System.out.println ("Character Counter");
      System.out.println ();

      // Read in a string and find its length
      System.out.print ("Enter a sentence or phrase: ");
      phrase = scan.nextLine();
      length = phrase.length();

      // Initialize counts
      countBlank = 0;
```

```
      // a for loop to go through the string character by character
      // and count the blank spaces

      // Print the results
      System.out.println ();
      System.out.println ("Number of blank spaces: " + countBlank);
      System.out.println ();
    }
}
```