PARAMETER PASSING
and
Methods which RETURN a VALUE

/* file == CollatzGeneric.java

RECALL: Collatz.java from Unit 2.
The positive integer \( n \) is changed either to \( n/2 \) or \( (3n+1)/2 \)
depending on whether \( n \) is odd or even.

In the previous version,
each object
had an instance variable
and an instance method.
The instance method modified the instance variable.

In this version, we will have a function (method) which is
- passed an integer,
- processes the integer,
- returns a new value.

Here is the heading of the method
public int nextN ( int n )

public                  <-- Method will be accessible
int              <-- Method will RETURN an integer value.
(int n)  <-- Method MUST be given an integer
(see Calling line)

RULE: The CALL to the function must match the function's HEADING.
If the heading says it "expects" an integer, then you must pass it an integer.

Here is a Call to the method
n = c.nextN(n);

The value returned by the method is assigned to the variable \( n \)

RULE: If a function is defined to return some type - then it must contain at least one
return statement (and it better be of the correct type.)
*/
**Question: Is this better?**

In a small class, as most of our examples are, it hardly matters.

However, you should imagine a large class which contains many functions. For example, you might want to have a class called PayCheck which calculates the paycheck of various employers of a large corporation. This class could have many different methods/functions to calculate the paycheck for various employees: calcManager(), calcAssistantManager(), calcLevel1Employee(), calcLevel2Employee(), etc.

Each of these need their own and different variables.

```java
public class PayCheck {
    private double rateForManagerDepart1,
                    rateForManagerDepart2,
                    rateForAssistantManager,
                    rateForLevel1,
                    rateForLevel2,
                    rateForLevel3,
                    etc;

    private double calcManager(char department) {
        
    }

    private double calcLevel1 ( double hoursWorked ) {
        
    }

    etc

    For example 1, the managers of different departments may be payed at different rates. And the method needs to know which department. (This could be indicated by a single letter.)

    For example 2, there might be many levels of hourly employees each getting a different payrate. In addition, each employee may work a different number of hours a week.

    Are you going to have one variable for them all to work on? or are you going to have different variables for each? Either way, you are adding extra overhead work for the programmer to worry about in order to set the correct variable for the function.

    By "passing the variable" to the function, you don't need to worry about which variable to use.

    This is why many of the predefined functions in Java (and other languages) require you to pass information (data, values, variables) to a method. For example, the rounding function is a method in the Math class Math.round(x). In order to use it, we must "tell it" the number it should "round".
```
import e50utils.*;

public class CollatzGeneric
{
    private int coef = 3;

    CollatzGeneric () { ; }
    CollatzGeneric ( int qqq )
    {
        if ( (qqq < 1) || ((qqq % 2) == 0) )
        {
            System.out.println("oops - Integer must be " + "positive and odd.");
            System.exit(0);
        }
        coef = qqq;
    }

    public int nextN ( int n )
    {
        if ( n % 2 == 1 )
            return (coef * n + 1)/2;
        else
            return n/2;
    }

    public static void main (String args[])
    {
        System.out.print( "Input a positive integer to multiply by: " );
        int ccc = SavitchIn.readLineInt() ;

        CollatzGeneric c = new CollatzGeneric();
        CollatzGeneric d = new CollatzGeneric( ccc );

        int k = 17, n = k;
        System.out.println("\tLet's see which reaches 1 first" );
        System.out.println("*" + 3 + "\t" + ccc);
        System.out.println("---\t---");
        System.out.println(n + "\t" + k);
        while ( n > 1 )
        {
            n = c.nextN(n);
            k = d.nextN(k);
            System.out.println(n + "\t" + k);
        }
    }
}
file == MarchingAnts.java

In this class, there are two methods besides the main().
Each is called 3 times.

**OBSERVE:** That each time we call the function "verse(#)" we are **passing it a different number.**

**NOTE:** Everything in this class is defined as static.
It is a standalone program-class which is unlikely to be used or extended by any other.

```java
class MarchingAnts
{
    static void verse (int n)
    {
        System.out.print("The ants came marching "
                        + n + " by " + n
                        + ", hurrah! Hurrah!\n"
                        + "The ants came marching " + n + " by "
                        + n + ", hurrah! Hurrah!\n"
                        + "The ants came marching " + n + " by "
                        + n + ",\n" + "The little one stopped to ");
    }
    static void chorus ()
    {
        System.out.print("And they all go marching down.\n"
                        + "To the ground. To get out of the rain!\n"
                        + "(Boom, boom, boom ... )\n"
                        + "\n");
    }
    public static void main (String[] args)
    {
        verse(1);
        System.out.println("suck his thumb.");
        chorus();
        verse(2); System.out.println("tie his shoe, "); chorus();
        verse(3); System.out.println("climb a tree, "); chorus();
    }
}
```

**VOCABULARY:**

- **static void verse (int n)**  
  The "n" is referred to as a **parameter** of the function. 
  From the example, we see that it can be used inside 
  the function. It is a **variable** of the function.

- **verse(1)**  
  When we call the function, we passed an actual 
  number to the function. The number "1" we pass is 
  referred to as an **argument**.
file == SameName.java

In Java, two methods in the same class may have the same name so long as their argument lists are different. The compiler does not look at the return type. When there is a single function with a given name, Java can automatically cast, but only in the more complicated direction. That is an integer can be automatically cast as a float but a float will not be automatically cast as an integer.

The method sameName() is said to be OVERLOADED.

class SameName
{
    int sameAsOther( int a, int b)
    {  return a + b;  }

    float sameAsOther ( float a, float b )
    { return a * b; }

    char sameAsOther ( int a, double b )
    {
        if ( a < b ) return 'M';
        else return 'Q';
    }

    double thisIsDifferent ( double a, double b )
    {
        System.out.println( "The first number is: " + a
                         +" and the second number is: " + b
                          );
        return 3.14;
    }
}

public static void main (String args[])
{
    int n = 1, m = 2;
    float x = 1.0f, y = 2.0f;

    SameName sn = new SameName();

    System.out.println( sn.sameAsOther( n, m ) );
    System.out.println( sn.sameAsOther( x, y ) );
    System.out.println( sn.sameAsOther( n, 10.0 ) );
    System.out.println();
    System.out.println( sn.thisIsDifferent( y, n ) );
}

/* OUTPUT
3
2.0
M
The first number is: 2.0 and the second number is: 1.0
3.14
*/
/* file == AfterB.java

PROBLEM: A secret message is hidden inside a long string of characters. To read this message, you must strip away all the characters except the ones immediately following the letter b. Of course this means you must worry about both the capital and lower case.

Method ANALYSIS
We want a method which receives a string.
Strips away the unwanted letters.
And returns another string.

So the heading of this function is: String decode ( String s )

Sub-PROBLEM: How do we strip away the unwanted letters.

loop through the input string
one character at a time.
If the current character is 'b' or 'B'
then save the next character.

RECALL: String is a predefined class in java. As such it comes with many methods.
This program illustrates .length() and .charAt(int)
Both of these return a value. .length() returns an integer, .charAt(i) returns a character. The second must be passed an integer -.charAt(i) must be passed an integer.

OBSERVE: The boolean condition using or "||".

class AfterB
{
    static final String TEST_MESSAGE = "YPBIUEB fBsibtuObiNBlYb1E0 kbMhb1GkBsaBswB mBKABaaZbrevbewUblgB."

    static String deCode( String s )
    {
        String temp = "";
        for ( int i = 0; i < s.length() - 1; i++ )
            if ( s.charAt(i) == 'b' || s.charAt(i) == 'B' )
                temp = temp + s.charAt(i+1);
        return temp;
    }

    public static void main (String args[])
    {
        System.out.println( TEST_MESSAGE );
        System.out.println( deCode(TEST_MESSAGE) );
    }
}
RANDOM NUMBERS:

/* file == RandomNotB.java

RECALL: In the decoding problem AfterB.java,
        we had a message "hidden" after the letter 'b'.

Naturally, there should be a program to create the "coded message". (run MakeAfterB.java)

Here is a Sub-Problem from that program.

PROBLEM: Write a method which returns a random letter of the alphabet
         - except for the letter B.

         Since we are not passing anything to the method, here is its heading:

            char randomCapitalNotB ()

SUB-PROBLEM: How do we get something "random".
Answer: Java has a predefined method in the Math class.

Math.random()        <-- returns a "pseudo" random number in the
range [0,1) which means:
                     Greater than or equal to 0
                     and strictly less than 1.

DIGRESSION: We won't get into how this function works -
            but we do want to mention that random-number-generators
            are extremely important for computer simulation of "real-world"
            events, and is an active area of research.
SUB-PROBLEM: How do we convert the output of Math.random() into a letter - and make sure it is not 'B'?

For Simplicity, we will just work with capital letters.

FACT: The capital letters are encoded (ASCII) as the integers 65=A .... 90=Z (see ShowAscii.java)

Math.random()            <--- is between 0 and 1
                        0 included, 1 excluded.
                        It's type is double.

26 * Math.random()      <--- is between 0 and 26
                        0 included, 26 excluded.

(int)(26*Math.random()) <--- CASTS this as an integer
                        range is: 0 --> 26
                        0 included, 26 excluded

(int)(26*Math.random()) + 1 <--- is an integer
                        range 1 --> 26
                        both 1 and 26 included.

(int)(26*Math.random()) + 65 <--- is an integer in the
                        range 65 --> 90
                        both 65 and 90 included

(char)((int)(26*Math.random())+65) <--- CASTS this an a character.
                        Using the ASCII encoding
                        65 --> A
                        90 --> Z

SUB-PROBLEM: Now we have an expression, which will give us a random capital letter. We still need to make sure it is not B.

ANALYSIS:
Check if it is B.
If it is, get another random capital letter until we get one that is not B.

*/
public static void main (String args[]) {
    for ( int i = 0; i < 30; i++ )
        System.out.print( randomCapitalNotB() + " ");
    System.out.println();

    System.out.println("\n\n\tThis one might have B in it.");
    for ( int i = 0; i < 30; i++ )
        System.out.print( (char)((int)(26*Math.random())+65) + " ");
    System.out.println();
}

/*OUTPUT
/unit3 % java RandomNotB
P T G C I Z I V Z O J F Q D O K G D Q E N Q M U C X Y D E Z

This one might have B in it.
/unit3 %*/
Random Numbers

file == RanDemo.java

**PROBLEM:** Simulate the rolling of a die.

* unit 3 lecture notes
* Chris Morrison, Dr. Henry H. Leitner
* Last Modified: July 1999

```java
import utils.*;

class Randemo {
    public static void main (String[] args) {

        OutFmt.setDecimalPlaces(3);
        System.out.println(" ROLL DEM BONES!
                           RANDOM BATTING AVERAGES");

        for (int outer = 1; outer <= 10; outer++)
        {
            for (int inner = 1; inner <= 6; inner++)
                System.out.print (OutFmt.RJust(OutFmt.fmt(
                    (int)(Math.random() * 6 + 1), 3));

            System.out.print (" ");

            for (int inner = 1; inner <= 6; inner++)
                System.out.print (OutFmt.RJust(OutFmt.fmt(Math.random(), 6) ) );

        System.out.println();

    }
}

/* OUTPUT
/unit3 % java RanDemo
 ROLL DEM BONES! RANDOM BATTING AVERAGES
 3 4 5 1 3 3  0.860 0.818 0.560 0.866 0.211 0.698
 5 4 3 5 5 4  0.743 0.800 0.084 0.358 0.224 0.218
 4 3 4 3 5 5  0.560 0.801 0.257 0.816 0.405 0.526
 6 6 3 5 3 3  0.064 0.397 0.789 0.189 0.084 0.123
 1 5 5 3 1 2  0.020 0.354 0.805 0.395 0.445 0.272
 2 5 6 3 1 4  0.843 0.187 0.744 0.252 0.185 0.065
 3 3 2 5 3 5  0.181 0.874 0.803 0.115 0.061 0.639
 3 4 4 4 5 3  0.977 0.832 0.419 0.194 0.862 0.577
 6 1 5 5 1 2  0.861 0.709 0.744 0.195 0.015 0.649
 2 4 4 4 1 4  0.301 0.996 0.153 0.077 0.055 0.143
```
Booleans Revisited

PROBLEM: Write a method which determines if an inputted integer is between 1 and 10 inclusive.

PROBLEM: Write a method which determines if an inputted integer is either strictly bigger than 10 or less then or equal to -10.

class OneTenAway
{
    static final int ONE = 1, TEN = 10;

    boolean oneToTen ( int n )
    {
        if ( ONE <= n && n <= TEN )
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    boolean tenAway ( int n )
    {
        if ( n <= -TEN || TEN < n )
        {
            //else
        }
        return true;
    }
}

public static void main (String args[])
{
    OneTenAway ota = new OneTenAway();
    System.out.println("Number" + "	1->10" + "	-10->10");
    for( int k = -10; k <= 15; k = k+5 )
    {
        System.out.println( k + "	" + ota.oneToTen(k) + "	" + ota.tenAway(k) );
    }
}

java OneTenAway
Number 1->10 -10->10
-10 false false
-5 false true
0 false true
5 true true
10 true true
15 false false
PRECEDENCE: Why are there not more parenthesis in the expressions in the previous programs?
That is, why not use \( \text{if (} (n < -10) \; || \; (10 < n) \; \text{)} \)?

ANSWER: The \textit{relational operators have a higher precedence} than the logical connective.

\[
n < -10 \quad \text{-- Returns a boolean value.} \\
< \quad \text{-- Is a relational operator.} \\
\quad \text{The two operands on either side should be numbers} \\
\quad \text{(char(acters) may also be compared).}
\]

\[
|| \quad \text{-- Is a logical operator.} \\
\quad \text{The two operands on either side should be booleans.}
\]

COMPARE: multiplication * has higher precedence than addition +
\[
(5 + 2 * 6 = 17)
\]
So, the * is done before the +
Likewise:
the < is done before the ||.

Note: There is nothing wrong with adding the parenthesis "to be sure".
ILLUSTRATION: There is always more than one way to get a Boolean Condition acting the way you want. Two Boolean Conditions are **EQUIVALENT** if they give the same value of true and false in all cases.

```java
class Logical {
    public static void main (String args[]) {
        System.out.println( "i\tj\ti==j\t!(i==j)\ti>j||j>i" );
        for ( int i = 0; i <= 1; i++ )
            for ( int j = 0; j <= 1; j++ )
                System.out.println( i + "\t" + j + "\t"
                    + (i==j) + "\t"
                    + !(i==j) + "\t"
                    + ( i < j || j < i ) );
    } //endOfmain()
} //endOfClass

/* OUTPUT
java Logical
i       j       i==j    !(i==j) i>j||j>i
0       0       true    false   false
0       1       false   true    true
1       0       false   true    true
1       1       true    false   false
*/
```
/*PROBLEM: Read the price of gold on 5 consecutive days, print the average price, and whether or not the price increased each day.

SUB-PROBLEM: How do we "remember" whether yesterday’s price increased over the price two days ago?

ANSWER: We will use a "boolean variable" to contain the fact that the price is increasing or not.

What does the program need to remember?
- today’s price
- yesterday’s price
- the sum (a running sum)
- whether or not the price increased

VARIABLES NEEDED
- a real-variable == float for today’s price
- a float for yesterday’s price
- a float for the running sum
- a boolean
- and an integer for the for-loop

ANALYSIS
- initialize the variables
- loop 5 times
  - read the next price
  - update the sum
  - update whether or not increasing
- output final answer

Note the usage of comments to explain the variables.

Compare the update of sum with the update of increasing.

/*
ILLUSTRATES: a method which returns a boolean.
The method accesses instance variables.
*/
import e50utils.*;

class Gold {
    int N = 5; // Number of days to use
    double oldPrice; // yesterday's price
    double price = 0.0; // today's price; it's zero because .. we haven't read any yet
    double sum = 0.0; // sum of all prices, initialized

    public boolean isIncreasing() {
        boolean increasing = true;
        // boolean value if price is increasing
        // assume true to start with
        System.out.print("\nType a double value that ");
        System.out.print("represents the price of gold...\n");

        for (int day = 1; day <= N; day++) // for each of n days ...
        {
            oldPrice = price; // save yesterday's price
            System.out.print ("On day #" + day + ": \$");
            price = SavitchIn.readLineDouble(); // read today's
            sum += price; // update price
            // An idiot-proof version would allow user to verify price
            if (price < oldPrice)
                increasing = false;
        }

        System.out.print ("Average price was ... \$");
        System.out.println(OutFmt.RJust(OutFmt.fmt(sum/5.0),2));

        return increasing;
    }
}
public static void main (String[] args)
{
    Gold g = new Gold();
    Gold h = new Gold();

    if ( g.isIncreasing() ) // if increasing is true
        System.out.print ("Price INCREASED every day!\n");
    else
        System.out.print ("Price DID NOT increase
                          every day!\n");

    System.out.println( "\tHere are the sums for
                        'g' and 'h'");
    System.out.println( "\tg.sum = "$ +        
                        OutFmt.RJust(OutFmt.fmt(g.sum),2) +   
                        "\th.sum = "$ + OutFmt.RJust(OutFmt.fmt(h.sum),2) );
}
} // end of class
Value vs Reference

Primitive Types vs Class Types

Why do class types require "new" and primitive types do not?

AnObject O; <- Declares the variable O and its type.
O = new Object() <- Allocates Memory and Initializes variable O

What does this mean when we pass the two different types to a method?

RECALL:

char ch = 'A'; <- creates a variable-box named ch and puts the letter 'A' in it.
int i = 1; <- creates a variable-box named i and puts the number 1 in it.

String s = "one" <- creates 2 variable boxes. The first is named s and contains the ADDRESS of the second box which contains "one"

char ch = 'A';

\[ \text{char} \ ch = \ 'A' ; \]

\[ \begin{array}{c}
    \text{ch} \\
    \text{ch} \\
    \end{array} \]

int i = 1;

\[ \begin{array}{c}
    \text{i} \\
    \end{array} \]

String s = "one";

\[ \begin{array}{c}
    \text{s} \\
    \text{s} \\
    \end{array} \rightarrow \begin{array}{c}
    \text{one} \\
    \end{array} \]

String name;

\[ \begin{array}{c}
    \text{name} \\
    \end{array} \]
FACT: When we pass an integer to a method, the parameter gets the \textbf{VALUE} of the integer.

FACT: When we pass an object to a method, the parameter gets the \textbf{ADDRESS} of the object.

RULE: A method can modify the value of its incoming parameter - but \textbf{CANNOT} modify the value of the original (primitive) argument.

\begin{verbatim}
    changeInts(m,n);       \hfill <-- can\textbf{NOT} change m or n
    .....a *= 2;            \hfill <-- a does change
\end{verbatim}

RULE: A method can modify the \textbf{FIELDS} of the object it has the address of.

\begin{verbatim}
    changeDemos(x,y);     \hfill <-- a gets same address as x
    a.n = 3;              \hfill <-- changes original x.n
\end{verbatim}

class ModifyParameters
{
    static \textbf{int} n = 2;

    static void showInts( \textbf{int} a, \textbf{int} b )
    { System.out.println( "First = " + a + "\tSecond = " + b ); }

    static void showDemo( DemoOnly a, DemoOnly b )
    { System.out.println( "\tFirst = " + a.n
                        + "\tSecond = " + b.showYourSelf() ); }

    static void changeInts( \textbf{int} a, \textbf{int} b )
    { a *= 2;  b %= 2; showInts(a,b); }

    static void changeDemos( DemoOnly a, DemoOnly b )
    { DemoOnly A = new DemoOnly( 2 * a.n );
      DemoOnly B = new DemoOnly( b.n \% 2 );
      a = A;  b = B;
      showDemo(a,b);
    }

    static void reallyChange( DemoOnly a, \textbf{int} b)
    { a.n = 3;
      b = 6;
      n = 8;
    }
}
public static void main (String args[]) {  
    int m = 1;
    showInts(m,n);
    changeInts(m,n);
    showInts(m,n);

    DemoOnly x,y;
    x = new DemoOnly();
    y = new DemoOnly(5);
    showDemo(x,y);
    changeDemos(x,y);
    showDemo(x,y);

    System.out.println("\n\n");
    reallyChange(x,m);
    showInts(m,n);
    showDemo(x,y);
}

//This is the class
class DemoOnly
{
    int n;
    DemoOnly(){n=-9;}
    DemoOnly(int i){n=i;}
    int showYourSelf(){return n;}
}

/*OUTPUT
java ModifyParameters
First = 1       Second = 2                <-- Before changeInts
First = 2       Second = 0                <-- Inside changeInts
First = 1       Second = 2                <-- After  changeInts

First = -9      Second = 5        <-- Before changeDemo
First = -18     Second = 1        <-- Inside changeDemo
First = -9      Second = 5        <-- After  changeDemo

First = 1       Second = 8                <-- m,n
First = 3       Second = 5                <-- x,y After reallyChange
*/

Observe The class DemoOnly has 2 constructors


DISCUSSES: Swapping variables.

Common Problem You have two variables. You want to put them in "order"
Solution: compare them.
If they are out of order: Swap Them.

SUB-PROBLEM: How do we SWAP the contents of two variable boxes?
Analysis: see the actual methods below.

Versions presented:
*  *
*  three that successfully work with class instances:
*  - member fn, one arg and receiver
*  - member fn, two args (redundant: ignores receiver)
*  - non-member, two args
*  *
*  one that tries to exchange two int variables - doesn't work.
*  *

WARNING: - java's wrapper classes don't help in this context, because they're immutable!
*  *
*  Chris Morrison, Dr. Henry H. Leitner
*  Last Modified: August 1999
*/

class MyInteger  // just for making little objects to play with!
{
    int num;

    MyInteger (int n)  // constructor
    {
        num = n;
    }

    void swap (MyInteger another)  // Works: preferred. Swap has
    {                             // two "args" (1st is receiver)
        int temp = num;
        num = another.num;
        another.num = temp;
    }

    void swap (MyInteger x, MyInteger y)  // Works: but fails to
    {                                     // exploit receiver as "arg"
        int temp = x.num;                // to member fn.
        x.num = y.num;                   // This is not 'OOP'y
        y.num = temp;
    }
}//end of classMyInteger
import java.util.

public class SwappingVariables // the "main" class
{
    static void swap (MyInteger x, MyInteger y) // Works
    {
        int temp = x.num;
        x.num = y.num;
        y.num = temp;
    }

    static void swap (int m, int n) // Does not work
    {
        int temp = m;
        m = n;
        n = temp;
    }

    public static void main (String[] args)
    {
        MyInteger a = new MyInteger(5);
        MyInteger b = new MyInteger(2);
        int num1 = 17;
        int num2 = -22;

        System.out.println ("a's value: " + a.num + '	' + "b's value: " + b.num + '
');
        a.swap (b);
        System.out.println ("a.swap (b) gives	" + "a.num: " + a.num + '	' + "b.num: " + b.num);
        a.swap (a, b);
        System.out.println ("a.swap (a, b) gives	" + "a.num: " + a.num + '	' + "b.num: " + b.num);
        swap (a, b);
        System.out.println ("swap (a, b) gives	" + "a.num: " + a.num + '	' + "b.num: " + b.num);

        System.out.println ("\nnow try swapping primitive int variables: ");
        System.out.println ("\nnnum1 & num2 are " + num1 + ", " + num2 + '
');
        swap (num1, num2);
        System.out.println ("after swap (num1, num2), num1 & num2 are still " + num1 + ", " + num2 + '
');
    }
}
/* OUTPUT
is03:~/unit3 % java SwappingVariables

a's value: 5    b's value: 2

a.swap (b) gives a.num: 2    b.num: 5
a.swap (a, b) gives a.num: 5    b.num: 2
swap (a, b) gives a.num: 2    b.num: 5

now try swapping primitive int variables:

num1 & num2 are 17, -22

after swap (num1, num2), num1 & num2 are still 17, -22

*/

/*

We are interested in writing our own functions (methods), for our own classes. When we do, we should ask:

"What information does it need?"
"What variables will it change?"
"What value does it return?"

DIGRESSION:
Some functions PERFORM CHORES (i.e. they are really "procedures")
Some functions make a calculation and return the result.

This function will not return anything : void myfunctionone ()
This function will return an integer int myfunctiontwo ()
This function will return a double double myfunctionthree() 

This function receives nothing: int myfunction4 ()
This function receives an integer: int myfunction5 (int n)
This function receives two integers: int myfunction6 (int m, int n)
This function receives a character and an integer: int myfunction7(char c, int n)
**PROBLEM:** Write a function that is passed two parameters a character and an integer. The program should write out the character the integer number of times.

**ANALYSIS:**

- Loop the integer number of times
- print out the character

**Observation:** The function receives two parameters - but does not change either.
**Observation:** The function is doing something - not returning a value.

**VARIABLES NEEDED** for the function:
- a parameter to receive the incoming integer
- a parameter to receive the incoming character
- a local integer to use in the for-loop

**JUST FOR FUN,** we will add an extra variable and permit the method to skip a designated number of spaces between each character.

**RE-ANALYSIS:**

- Loop the integer number of times
- Loop the number of spaces -- printing a space each time
- print out the character

Here is the Header for the function definition

```java
void nwrite ( char ch, int nrep, int nspace)
```

There are **three parameters** to this function: a character, an integer, an integer. When we call the function - we must **pass "matching"** arguments in the same order.
NOTE: The function has 4 LOCAL-VARIABLES ...... 3 of them are parameters. No other function can use these variables. Although, other functions could have local variables with the same names.

NOTE: In calling the function, we are actually passing a variable and a 2 literal values.

OBSERVE: for loop nested inside of a while loop. This could have been done with two for-loops.

```java
import utils.*;

class GraphMaker
{
    void nwrite ( char  ch, int nrep, int nspace )
    {
        while ( nrep > 1 )
        {
            System.out.print (ch);
            for ( int  j =1; j <= nspace;  j++)
            {
                System.out.print (' ');
                nrep --;
            }
            System.out.println (ch);
        }
    }
}
```
void make3 ()
{
    int production;
    for (int year = 1998; year <= 2000; year++)
    {
        System.out.print ("Enter production figures for England, 
                        France and Japan");
        System.out.println ("& Japan in " + year + " on one line:\n");
        for (int counter = 1; counter <= 3; counter++)
        {
            production = SavitchIn.readInt();
            System.out.println();
            switch (counter)
            {
                case 1: nwrite ('*', production,  2); break;
                case 2: nwrite ('$', production,  0); break;
                case 3: nwrite ('%', production,  1); break;
                default: System.out.print ("THIS SHOULD NEVER HAPPEN;
                                            WHY NOT?");
            } // end of switch statement
        } // end of inner for loop
        System.out.println (year + "\n\n");
    } // end of outer for loop
    System.out.println();
    System.out.println ("Key to symbols: * is England $ is France % is Japan ");
}

public static void main (String[] args)
{
    GraphMaker gm = new GraphMaker();
    gm.make3();
    gm.nwrite ('A',35,0);
}

OBSERVE: uses the counter of the for-loop inside the switch statement.
PROBLEM: We need to input some numerical grades and convert them into letter grades - or in this case some remarks.

ANALYSIS: A large nested if else statement

```java
    if grade < 5 then fail
    else if grade < 7 then not so good
    else if grade < 8.5 then decent
    else if grade < ......
```

This quickly can become quite long and not always that easy to follow. So we use the SWITCH statement instead.

ILLUSTRATES: `switch` statement.

WARNING: The "break" statement is necessary, otherwise the case will "fall" into the next case.

(see PickUpSticks.java)

```java
    case 1:  case 2:     ButtonShow(); break;
    case 3:  case 4:     CloseDoor();
    case 5:  case 6:     pickUpSticks(); break;
```

In this situation, a value of 3 will both CloseDoor() and pickUpSticks().

ILLUSTRATES: a `sentinal value` indicating no more data.

ILLUSTRATES: default: for switch statement

ILLUSTRATES: "random" order of cases
import utils.SavitchIn;

class Exams
{
   public static void main (String[] args)
   {
      int examScore = 0;

      while (examScore != -1)
      {
         System.out.print ("INPUT A STUDENT'S EXAM SCORE,\n");
         System.out.print (" OR -1 TO END THE LOOP: ");
         examScore = SavitchIn.readLineInt();

         switch (examScore)
         {
            case -1:  // end of input
               break;
            case 7:
            case 6:
               System.out.print ("Barely Passing...\n\n");
               break;
            case 3:
            case 4:
               System.out.print ("Flunking\n\n");
               break;
            case 2: case 0: case 1:
               System.out.print ("Exceptionally Flunking\n\n");
               break;
            case 8:
            case 9:
               System.out.print ("Good\n\n");
               break;
            case 10:
               System.out.print ("Exceptionally Good!\n\n");
               break;
            case 5:
               System.out.print ("Barely Passing\n\n");
               break;
            default:
               System.out.print ("Not a legal grade!\n\n");
               break;
         } // end of switch block
      } // end of while statement
      System.out.print ("No more data!\n");
   }
}
file == Combin.java

**PROBLEM:** Write a function which accepts two positive integers i and n and returns the number of distinct subsets of i objects selectable from n objects.

This function, receives two integers which it does not change and returns an integer value to the calling function.

Here is the header

```
int combin(int, int);
```

We will use a well known formula

\[
\binom{n}{i} = \frac{n \cdot (n-1) \cdot (n-2) \cdots \cdot (n-i+1)}{1 \cdot 2 \cdot 3 \cdots \cdot i}
\]

**DIGRESSION:** You are being given an *algorithm* in the form of a *formula* - which you may or may not understand. HOWEVER, you should understand what it is telling you to do and you should be able to write a program from it.

**Warning:** the obvious way to program this would be to multiply out the top, multiply out the bottom and then do the division.

```java
for ( int t = n; t >= n-i+1; t-- ) top = top * t;
for ( int b = 1; b <= i; b++ ) bottom *= b;
return top/bottom;
```

This would very quickly overflow. (Demo CombinTopBottom.java: input C(20,10))

Instead, we will calculate by multiplying by a term on the top and then dividing by a term on the bottom.

```
temp = n * (n-1) * ... * (n-count+1) / count
```
ILLUSTRATES the **do-while** loop.

```java
import e50utils.*;

class Combin
{
    int combin (int n, int i)
    {
        int temp = 1, count = 0, mult = n;

        // INVARIANT FORM for temp:
        // temp = n * (n-1) * ... * (n-count+1) / count!

        if ((i < 0) || (i > n))
            return 0;
        else
        {
            while ( count < i )
            {
                count ++;
                temp = temp * mult / count;
                mult -- ;
            }

            return temp;
        }
    }
}
```
```java
public static void main (String[] args)
{
    Combin c = new Combin();

    int nObjects, subset;
    char answer;

    do
    {
        System.out.print ("How many objects in the set? ");
        nObjects = SavitchIn.readLineInt();

        System.out.print ("How many in the subset? ");
        subset = SavitchIn.readLineInt();

        System.out.println ("There are " +
            c.combin (nObjects,subset) + " combinations!");

        System.out.println ("Want to do this some more? ");
        answer = SavitchIn.readLineNonwhiteChar();
    } while (answer == 'y' || answer == 'Y');

    Combin() // a constructor, snuck in at the bottom of the class.
    {
        System.out.println
            ("---Welcome to the world of Combinatorics.");
    }
}

/*OUTPUT
is01:~:/unit3 % java Combin
---Welcome to the world of Combinatorics.
How many objects in the set? 10
How many in the subset? 2
There are 45 combinations!
Want to do this some more?
yes I do
How many objects in the set? 45
How many in the subset? 9
There are -68274041 combinations!
Want to do this some more?
n
*/
```
file == CubeRoot.java

**PROBLEM:** Calculate the cube root of an arbitrary positive number.

This program will calculate the CUBE-ROOT of any positive number, N, using the "Newton-Raphson" **algorithm.** This is an existing algorithm - we want to translate the algorithm into a program.

The basic idea is:

- make a guess
- loop until a predetermined accuracy is attained
- improve the guess using the known formula.

We make a guess at the cube-root and then **IMPROVE** the guess by the update

\[
guess = \frac{2 \times \text{guess} + \frac{\text{the_number}}{\text{guess} \times \text{guess}}}{3}
\]

**VARIABLES:**

- The function is going to be passed 2 doubles (the number and the guess)
- and return a double.

**ADDITIONALLY:** In dealing with real numbers, we do not look for exact answers but only answers within a prescribed accuracy. This example demonstrates a simple way to do this.

```java
import e50utils.*;

class CubeRoot {
    static double PRECISION = 0.1;

    static void setPrecision ( double p )
    { if ( 0 < p && p < 1 ) PRECISION = p; }

    static double nextGuess( double d, double guess )
    {
        return ( 2 * guess + ( d / (guess * guess) ) ) / 3;
    }
}
```
static double approxCubeRoot(double d, double guess) {
    int count = 0;
    do {
        guess = nextGuess (d, guess);
        count++;
        System.out.println( "guess is now " + guess + 
            "\n\t\t cubed it is " + guess*guess*guess + 
            "\n\t\t difference is " + (guess*guess*guess - d));
        e50Input. pauseLine();
    } while( Math.abs( guess*guess*guess - d ) >= PRECISION );
    return guess;
}

public static void main (String args[]) {
    approxCubeRoot(8, 1);
}

} //end of class

RECALL: Since everything in the above class is "static", it is easy for another class to access it.

class CubeRoot3 {
    public static void main (String args[]) {
        CubeRoot.setPrecision( 0.5 );
        double rootOfTen = CubeRoot.approxCubeRoot(10,2);
        System.out.println( "Just to test " + rootOfTen + " cubed is " + rootOfTen * rootOfTen * rootOfTen );
    }
}

/* OUTPUT
/unit3 java CubeRoot3
guess is now 2.1666666666666665
cubed it is 10.171296296296294
difference is 0.17129629629629406
.....Hit any the return key to continue....
Just to test 2.1666666666666665 cubed is 10.171296296296294
A SIDE EFFECT of a function can occur when a method effects non-local variables.

THOUGHT: When you think of a function as returning a value - you realize it probably shouldn't be changing anything.

```java
/** Sideff.java .................... run Sideff2.java
 * Unit 3 lecture notes
 * This example is taken from the DEC VAX Pascal Manual.
 * Chris Morrison, Dr. Henry H. Leitner
 */

class Sideff {
    static int x;
    static int y;

    static boolean positive (int thisvar)
    {
        if (thisvar > 0)
        {
            x = thisvar - 10;
            return true;
        }
        return false;
    }

    public static void main (String[] args)
    {
        y = 7;  x = 5;
        System.out.println ("ANS1 equals "
                + (positive(x) && positive(y))
        );

        y = 7;  x = 5;
        System.out.println ("ANS2 equals "
                + (positive(y) && positive(x))
        );
    }
}
/* OUTPUT
java Sideff
ANS1 equals true
ANS2 equals false
is03:~/unit3 % */
```
/ WARNING: NEVER COMPARE floats of doubles.

/**
* EqReal.java
* unit 3 lecture notes
* Shows the futility of testing for equality of DOUBLE values
* 
* Chris Morrison, Dr. Henry H. Leitner
* Last Modified August 10, 1999
*/

class EqReal
{
    public static void main(String[] args)
    {
        double x;
        for (x = 0.0; x != 1.0; x += 0.1)
        {
            System.out.println(x);
            if (x > 2) System.exit(0);
        }
        System.out.println("Final value of x is " + x);
    }
}
file == GradeCurver.java

Discussion: A computer science instructor wants to "curve" the grades of the students. The goal is that the lower grades should "curve" more than the higher grades. Also, a grade of 0 should stay a grade of 0 and a grade of 100 should remain a 100.

A common formula for this is

\[
\text{curved grade} = 100 \times \left( \frac{\text{grade}}{100} \right)^x
\]

where \( x \) is chosen between 0 and 1 (0 must be excluded) and varies from test to test.

PROBLEM: Create a class to implement this formula.

ANALYSIS: I want a function which has a grade passed to it and returns the curved grade. As an illustration, I will require that the function accepts a float but returns an int.

DIGRESSION: There are 2 constructors.
   The second invokes the first.
Recall: A constructor is called when memory is allocated for the actual object - Not just when the variable is declared.

\[
\text{this} \quad <\text{-- Refers to the current object (not the class)}
\]
\[
\text{this()} \quad <\text{-- Inside the second Constructor - It is calling the other constructor.}
\]

Rule: Such a Constructor invocation must be the first thing in a method.

OBSERVE: Each object has it's own exponent - and these may be different.

*/
class GradeCurver {
    float exponent;

    // 2 Constructors.
    GradeCurver() {
        exponent = 1.0f;
    }

    GradeCurver( double d ) //Better for users
    {
        this();
        if ( d <= 0 || 1 <= d )
            System.err.println("Warning: unacceptable exponent!");
        else
            exponent = (float)d;
    }
}

int Curve ( float f )
{
    return (int)( 100 * Math.pow( f/100, exponent) + 0.5);
}

public static void main (String args[])
{
    GradeCurver g = new GradeCurver();

    GradeCurver h = new GradeCurver( 0.65 );

    GradeCurver k = new GradeCurver( -0.5 );

    System.out.println(" ........Curves........");
    System.out.println("Grade" + "\t" + g.exponent + "\t" + h.exponent + "\t" + k.exponent);

    for ( int i = 0; i < 121; i = i + 10 )
        System.out.println( i + "\t" + g.Curve(i) + "\t" + h.Curve(i) + "\t" + k.Curve(i) );
}
PROBLEM: Estimate the value of PI

We will estimate the value of PI using "Monte-Carlo" simulation.

Probabilistic notion: Pick a "random" point in the unit square. The percentage of times this random point is inside the unit circle should be 1/4 of the value of PI because one-fourth of the circle is in the unit square.

ANALYSIS: pick two random numbers x,y between 0 and 1. This represents a point in the unit square. If the sum of their squares is less than (or equal) to 1, then the point x,y represents is in the unit circle.

Do the above in a loop and count the number of "hits" then multiple by 4 and divide by the size of the loop.

/*
import e50utils.*;

class Pi
{
    public static void main (String[] args)
    {
        int   n;       // # of pieces of chalk to be thrown
        int   count;   // # of pieces of chalk already thrown
        int   hits = 0; // # of pieces which land w/in the qtr.-circle
        double x, y;   // coordinates of a point in the quarter-square

        System.out.print ("This program will approximate
                          the value of PI\n\n");
        System.out.print ("How many pieces of chalk
                          should be thrown? ");
        n = SavitchIn.readLineInt();

        for (count = 1; count <= n; count++)
        {
            // Throw 1 piece of chalk into quarter-square...
            x = Math.random();
            y = Math.random();

            if (x * x + y * y <= 1)
                hits++;         // It landed in the quarter-circle!

            if (count % 200 == 0)
                System.out.println ("   Running after "+ count + " hits...");

        }
    System.out.print ("\nThe approximate value of PI is ..");
    System.out.println (OutFmt.RJust( OutFmt.fmt(4 * (double)hits / n),2));
    }
}
/* file == P236.java

PROBLEM: A childern's game has a spinner with three colors.
Half of it is red, One third of it is green and one sixth of it is yellow.
Assuming everything is fair, we expect that the spinner will land on red half of the
time, on green one third of the time and on yellow one sixth of the time.

Write a program to simulate the output of this spinner

THERFORE: We want a METHOD/Function which returns randomly
one of the three words: "RED", "YELLOW" or "GREEN"

Here is the Header - and somewhere in the body of the method there must be at least one
RETURN statement returning a String.

    static String randomRGY()
Probabilty Analysis

We don't want the three words with equal probability.
Look at the actual expression to understand the "probability".

Math Check: $1/2 + 1/3 + 1/6 = 3/6 + 2/6 + 1/6 = 6/6 = 1$.

******* SKIP on First Reading************
We will discuss Arrays in Unit 4
ARRAY DIGRESSION: This example, shows some useful ways to employ ARRAYS.

```
[   ]               <-- Square brackets to indicate Array
freq[1]            <-- Square brackets to access the
different members of the array.

static             <-- Because we are making everything in
                    this example static.
final              <-- Because these will NEVER change
                    in the program.

int[] freq = new int[3];  <-- There will be
                          3 integer fields.
                          freq[0], freq[1], freq[2]

char[] LETTERS3 = { 'R', 'G', 'Y' };  <-- There are
                                3 Character fields.
                                They are initialized:
                                LETTERS3[0] = 'R'
                                LETTERS3[1] = 'G'
                                LETTERS3[2] = 'Y'
```

*/
import e50utils.*;

class P236
{
    static double HALF  = 1.0/2.0,
         THIRD = 1.0/3.0,
        SIXTH = 1.0/6.0;

    static final char[] LETTERS3 = { 'R', 'G', 'Y' };
    static final int[] freq = new int[3];

    static void showFreq()
    {
        System.out.println( "\t\tHere are current counts." );
        System.out.println( "Red\tGreen\tYellow" );
        for(int i=0; i<3; i++)
            System.out.print( freq[i]  + "\t" );
        System.out.println();
    }

    static int randomOf3()
    {
        double r = Math.random();

        if ( r < HALF ) return 0;
        else if ( r < HALF + THIRD) return 1;
        else if ( r < HALF + THIRD + SIXTH ) return 2;
        //else

        return -1;
    }

    static String randomRGY()
    {
        double r = Math.random();
        if ( r < HALF ) return "Red";
        else if ( r < HALF + THIRD) return "Green";
        else if ( r < HALF + THIRD + SIXTH ) return "Yellow";
        //else
        return "Puce";
    }
}
```java
public static void main (String args[]) {
    System.out.print ("Here is the sum of our 3 numbers: ");
    System.out.println( HALF + THIRD + SIXTH );

    System.out.println("Here are some random colors.");
    for ( int i = 0; i < 5; i++ )
        System.out.print( randomRGY() + " ");
    System.out.println();

e50Input.pauseLine();
System.out.println("\n\n\nHere is a 'Long' run.");

    int temp;
for ( int i = 1; i <= 1200; i++ )
    {
        temp = randomOf3();
        freq[temp]++;
        System.out.print( LETTERS3[temp] );
        if ( i % 60 == 0 ) System.out.println();
    }

    System.out.println();
    showFreq();
}
```
file == SemesterAverage.java

/* PROBLEM: Thinking about designing a class. The semester average.
   A typical student takes 4 courses, some of which are 3 credits,
   some are 4 credits and some are 5 credits. In each course the student
   receives either an A worth 4 points, a B worth 3 points, a C worth 2
   points, a D worth 1 point or an F worth 0 points.
   Design a class structure to deal with this information and to
calculate the average for the semester.

Analysis: We want to do the following.
   ask and input the grades
   calculate the average
   print out the semester average

Variable Analysis: We want to be able to store the following information.
   The student's name -- a String of characters
   The 4 courses -- we can think of each course as another object

ILLUSTRATION: We just decided on an additional class to the one we are designing.
   We should analyze this additional class and how we will use it.

Course Object Variable Analysis
   The number of credits -- an int(eger)
   The letter grade of the course -- char(acters or int(eger)?
   (of course we might also want to know the name of the course
   but we want to simplify the example.)

WARNING: There are certain aspects of the original problem description
which are missing. For example, what happens if somebody gets an
incomplete in a course. Although the problem doesn't address this, we
should be thinking about it ---- AND, where in the program should this be handled?
   In this example, we are having the sub-class handle the
situation. Note that this illustrates a possible reason why we don't
want to be able to access the instance variables of the sub-class directly
-- we might not use the correct value

REMARK: We have put both classes inside the same file -
because we are thinking of one class as a subclass of the other.
When we compile the file it creates both class files.

Alternatively, we could have put the class aCourse inside it's own file
-- and labeled it public. This would correspond to thinking of it as
a class that many other classes would want access to.
*/
import e50utils.*;

//public       <-- As it is not in a separate file - can't be public
class aCourse
{
    private int numberOfCredits;
    private char grade;
    // 4 constructors
    aCourse ()
    {
        numberOfCredits = 4;
        grade = '*';
    }
    aCourse ( int incomingInteger)
    {
        numberOfCredits = incomingInteger;
        grade = '*';
        check();
    }
    aCourse ( char incomingChar)
    {
        numberOfCredits = 4;
        grade = incomingChar;
        check();
    }
    aCourse ( int incomingInteger, char incomingChar)
    {
        numberOfCredits = incomingInteger;
        grade = incomingChar;
        check();
    }
<-- The usual import line at the beginning

<-- If it were public it would have to be in a separate file
<-- The other class

<-- A default constructor - initializing the instance variables
Constructors have the same name as the class
and no return type (not even void).
One of them is called automatically when the object
is created (by the "new").

<-- Another Constructor
There are 4 constructors
They have different parameter lists

<-- A third constructor
This is the one we are using in this example.

<-- A fourth constructor
private void check()
{
    if ( numberOfCredits < 3 || 5 < numberOfCredits )
        numberOfCredits = 4;

    grade = Character.toUpperCase(grade);

    switch ( grade )
    {
        case 'A': case 'B': case 'C':
            case 'D': case 'F':
            case 'E': case 'I': case 'W':
            case '*':   break;
            default : grade = '*';
    }
}

public int value()
{
    switch ( grade )
    {
        case 'A' : return 4 * numberOfCredits; //break;
        case 'B' : return 3 * numberOfCredits;
        case 'C' : return 2 * numberOfCredits;
        case 'D' : return 1 * numberOfCredits;
        case 'F' : return 0 * numberOfCredits;
        default  : return 0;
    }
}

public int creditHours()
{
    if( ('A' <= grade && grade <= 'F') && (grade != 'E'))
        return numberOfCredits;
    else return 0;
}

public char gradeInCourse()
{ return grade; }
}//end of class aCourse
A secret method of the class aCourse.
   It is called by the constructors (see them).
   It performs some "idiot proofing".

Calling a method in the predefined wrapper class

switch statement

These cases all do nothing.
   They "fall" onto the same break statement.

The default catches everything else,

A method which returns an integer.
   In this case, it is doing the work of calculating
   and handling cases like I and W etc.

Each case should end with a break-statement
   but we are cheating because we know the return-statement
   will automatically break and end the method.

A Method which returns an integer.
   It returns 0 for assorted grades such as I and W and *

A Method which returns a char(acter)
public class SemesterAverage
{
    private String name;
    private aCourse[] course = new aCourse[4];

    public double theAverage()
    {
        double totalCredits = 0.0, totalValue = 0.0;
        for( int i = 0; i < 4; i++ )
        {
            totalCredits += course[i].creditHours();
            totalValue += course[i].value();
        }
        return totalValue/totalCredits;
    }

    public static void main (String args[])
    {
        SemesterAverage sa = new SemesterAverage();

        int i; //Loop Index
        for( i = 0; i < 4; i++ )
        {
            System.out.print("Grade in course " + (i+1) + ": ");
            sa.course[i] = new aCourse( SavitchIn.readLineNonwhiteChar() );
        }
        System.out.println( sa.theAverage() );
    }
}
<-- The class we are designing and running

<-- An ARRAY of 4 objects of type aCourse

<-- An Instance METHOD
    It is RETURNING a value (to where it is called)

<-- There must be a RETURN line returning a value

<-- Creating an object
    Don't forget that sa is of type SemesterAverage.
    It has a variable called sa.course
    which is an array of 4 objects of type aCourse

<-- Creates object of type aCourse called sa.course[i] (i=0).
<-- Calling the CONSTRUCTOR which receives a single character.
    That is,
    the method readLineNonwhitechar (in class SavitchIn)
    returns a character (read from the keyboard)
    which is given to the proper constructor
    when the new object (of type aCourse) is created.
**Character Input**

**Streams of Characters**

**QUESTION:** Why do we need to use a special class to do input?

Java comes with two different systems to do input/output - and neither is "complete". The first is based on **Streams-of-bytes**. The second is based on **Streams-of-characters**. Internally a java program prefers Character Streams.

Another reason, is that Java was not originally intended to primarily communicate with a user at a keyboard. Often the input to a method comes from a different method in another object - or possibly from a form or button on a home page.

We will illustrate the various issues by examining the code from some of the files in the package e50utils.

```
System                           <-- A class in the package
java.lang
System.in                        <-- A Field of the class.
It's type is InputStream()
which is a class in the
package java.io
System.in.read()                 <-- A Method of the class
InputStream.
```

From the API:

```java
public abstract int read() throws IOException

Reads the next byte of data from this input stream. The value byte is returned as an int in
the range 0 to 255. If no byte is available because the end of the stream has been reached,
the value -1 is returned. This method blocks until either input data is available, the end of
the stream is detected, or an exception is thrown.

A subclass must provide an implementation of this method.

Returns:
the next byte of data, or -1 if the end of the stream is reached.

Throws
IOException (I-§2.29)
If an I/O error occurs.
```

System.in.read() reads the "next byte" and returns it as an "integer".
But we want to be able to read characters, integers and doubles. The programmer needs to
do the necessary conversions.
/* File==ReadFiveChar.java

**DEMO:** Reading 5 characters from the keyboard.
Printing them and their ASCII number.
**OBSERVE:** This will process the newline-character.
**OBSERVE:** On the UNIX machine <ctrl>d = eof = -1
**AND:** On the UNIX machine <ctrl>d needs to be on a separate line when we input it.

**ILLUSTRATES**
Array of integers - with initialization
Throwing an exception
*/

```java
class ReadFiveChar {
    public static void main(String[] args) throws java.io.IOException {
        int[] fiveInts = {-1,-1,-1,-1,-1};
        System.out.print("Give 5 characters: ");
        for( int counter = 0; counter < 5; counter++ )
            fiveInts[counter] = System.in.read();
        for( int i = 4; i >= 0; i-- )
            System.out.print("Character "+i+": "+fiveInts[i]);
    }
}
```

try inputting <ctrl>D

```
Give 5 characters: abcde
Character #4: 101 is = e.
Character #3: 100 is = d.
Character #2: 99 is = c.
Character #1: 98 is = b.
Character #0: 97 is = a.
```

```java
Give 5 characters: a b c
c
```

```
Character #4: 99 is = c.
Character #3: 10 is = 
Character #2: 98 is = b.
Character #1: 32 is = .
Character #0: 97 is = a.
```
From the package: e50utils

File == e50Input.java

This class "reads" from the keyboard.
* A separate class is used to convert an input string to
  * int or double.
**
* METHODS:
  * readChar()
  * readWord()
  * readLine()
  * readLine()
  * pauseLine();

ILLUSTRATES

package
Exception
try-catch block
Cast byte to char
End of file concerns
String Processing

package e50utils;

import java.io.IOException;

public class e50Input
{
  public static boolean EOF = true;

  public static char readChar()
  {
    EOF = false;
    int ch = -1;  //compiler requires initialization
    try
    {
      ch = System.in.read();
      if ( ch <= 0 ) { EOF = true; }
    }
    catch ( IOException e )
    {
      System.out.println( e.getMessage() );
      System.exit(0);
    }
    return (char)ch;
  }
}
public static String readLine()
{
    char ch;
    String s = "";

    ch = readChar(); if ( EOF ) return s;
    while ( ch != '\n' && ch != '\r' )
    {
        s = s + ch;
        ch = readChar();
        if ( EOF ) { EOF=false; return (s);}
    }
    return s;
}

public static String readWord()
{
    String s = "";
    char ch;

    ch = readChar(); //reads first char
    if ( EOF ) return s; // if EOF return s = ""

    while( Character.isWhitespace(ch) )
    {
        ch = readChar();if ( EOF ) return (s);
    }

    while( ! Character.isWhitespace(ch) )
    {
        s = s + ch;
        ch = readChar();
        if ( EOF ) { EOF=false; return s;}
            //don't want it false here.
            // return current word
            // EOF true next readchar
    }
    return s;
}//end of readWord()
public static void flushLine()
{
    try
    {
        System.in.skip (System.in.available());
    }
    catch ( IOException e )
    {
        e.printStackTrace();
        System.exit(0);
    }
}

public static void pauseLine()
{
    System.out.print( " Hit the return key to continue....." );
    char ch = readChar();
    flushLine();
}

}//end of class
DIGRESSION: Exceptions

/* EXCEPTIONS
   Throw
   try-catch

There are two kinds of RunTime Bugs that can crash
a program: Fixable and NonAsFixable.
(These are not official terms.)

Examples

Fixable: Dividing by zero.
   You could fix this by adding an if-else clause
to the program.

   if ( n = 0 ) System.out.println("Hey don't do that.");
   else System.out.println("Answer is: " + (y/x));

   Observe: This protects the program from crashing.
   In other words, it can go on and do the rest of its work.

NonAsFixable: You need more memory in your computer.
   (An advanced programming might analyze the gory details
   of memory use and figure out a way to rewrite the program.)
   You solve this by buying more memory - not rewriting the program.

When these occur, Java produces an Object.
   We calling this Throwing.

Java has two classes: Error and Exception.
   Error is for the second type.
   Exception is for the first type.

When an Exception object appears
   The method (where it occurred in the program)
can either catch and handle it
or throw it to a higher method
i.e the method which called the current one.

   If each method throws it higher
   it eventually reaches the Java Interpreter
   which says enough is enough and crashes the program.
DEMO: Run this to see what an error message looks like.

```java
import e50utils.*;

public class Exception0
{
    public static void main (String args[])
    {
        int n,m;
        System.out.print("Two integers please: ");
        n = e50Convert.toInt( e50Input.readWord() );
        m = e50Convert.toInt( e50Input.readLine() );
        System.out.print( n/m );
        System.out.println( "\t" + n + "/" + m );
    } //endOfmain()
} //endOfClass

/* OUTPUT - with a program crash
is04:~/unit4 % javac Exception0.java
is04:~/unit4 % java Exception0
Two integers please: 7 3
2       7/3
is04:~/unit4 % !
java Exception0
Two integers please: 7 0
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at Exception0.main(Exception0.java, Compiled Code)
is04:~/unit4 % */
/* file == Exception2.java

An advantage of this over just using an if-else clause
is that the error can be **handled at a higher level**
instead of only at the exact location it occurred.

**DEMO:** Illustrates the try-catch for errors.

*/
import e50utils.*;

public class Exception2
{
    public static void main (String args[])
    {
        int n,m;
        System.out.print("Two integers please: ");
        n = e50Convert.toInt( e50Input.readWord() );
        m = e50Convert.toInt( e50Input.readLine() );

        try
        {
            System.out.print( n/m );
        }
        catch ( ArithmeticException ae )
        {
            System.out.println( "\tException: "
                              + ae.toString() );
        }
        System.out.println( "\t" + n + "/" + m );
    } //endOfmain()
} //endOfClass

/* OUTPUT - Program didn't crash
is04:~/unit4 % javac Exception2.java
is04:~/unit4 % java Exception2
Two integers please: 7 _3
2       7/3
is04:~/unit4 % !!
java Exception2
Two integers please: 7 0
   Exception: java.lang.ArithmeticException: / by zero
   7/0
is04:~/unit4 % */
/*****
* File == e50Convert.java
*
* This class "converts" strings to the various primitive types.
*
* Part of the issue is that we will input "strings"
* from the keyboard.
*
* Note that this is a separate class from the Input-class.
* You should recognize that converting is a "separate-chore"
* from "reading-from-the-keyboard".
*
* METHODS:
*    toInt( String str )
*    toLong( String str )
*    toFloat( String str )
*    toDouble ( String str )
*    toBoolean( String str )
*
* EXPLANATIONS:
*    Integer.parseInt(str)    <-- Integer is the "wrapper class"
*                            <-- parseInt() is a
*                            Static member function,
*                            which receives a string.
*                            <-- It returns an "int"eger
*
*    (Slightly longer way to do)
*    Long.valueOf(str).longValue() <-- Long is the wrapper class
*                                <-- valueOf() is a
*                                Static member function,
*                                which receives a string.
*                                <-- It returns a Long object
*                                <-- .longValue() is a member function
*                                (not static)
*                                which returns a long
*    ******/
package e50utils;
//import java.io.IOException;

public class e50Convert
{
    public static int toInt ( String str )
    { return Integer.parseInt(str); }
        //Not Catching NumberFormatException

    public static long toLong ( String str )
    {
        long l = -9999; //some compilers need this.
        try { l = (Long.valueOf(str).longValue()); } 
        catch ( NumberFormatException e )
        { System.out.println( e.getMessage() );System.exit(0); } 
        return l;
    }

    public static float toFloat ( String str )
    {
        Float floatObj = new Float(str);
        return floatObj.floatValue();
    }

    public static double toDouble ( String str )
    {
        double d = -9999; //some compilers need this.
        try { d = (Double.valueOf(str).doubleValue()); } 
        catch ( NumberFormatException e )
        { System.out.println( e.getMessage() );System.exit(0); } 
        return d;
    }

    public static boolean toBoolean ( String str )
    {
        if ( str.equalsIgnoreCase("true") )
            return true;
        else return false;
    }
}
//end of class
package e50utils;

import java.text.*; // has DecimalFormat class

public class e50Format
{
    public static String lAlign( String s, int field )
    {
        String temp = "" + s;
        if ( temp.length() < field )
            temp = temp + plusSpaces( field - temp.length() );
        return temp;
    }

    public static String rAlign( String s, int field )
    {
        String temp = "" + s;
        if ( temp.length() < field )
            return rAlign(temp, field);
        return temp;
    }

    public static String lAlign( long l, int field )
    {
        String temp = "" + l;
        if ( l >= 0 ) temp = " " + temp; //space for "-"
        if ( temp.length() < field )
            return lAlign(temp, field);
        return temp;
    }

    public static String rAlign( long l, int field )
    {
        String temp = "" + l;
        if ( temp.length() < field )
            return rAlign(temp, field);
        return temp;
    }
}
// Two Methods to add spaces or zeros
private static String plusSpaces ( int howmany )
{
    String spaces = "";
    for ( int i = 0; i < howmany; i++ )
        spaces = spaces + " ";
    return spaces;
}

private static String plusZeros ( int howmany )
{
    String zeros = "";
    for ( int i = 0; i < howmany; i++ )
        zeros = zeros + "0";
    return zeros;
}

public static String doubleToString ( double d, int decPlaces)
{
    String pattern = "#0.";
    for ( int j = 0; j < decPlaces; j++ )
        pattern += '0';
    DecimalFormat df = new DecimalFormat ( pattern );
    return df.format(d);
}

public static String rAlign (double d, int field, int decPlaces)
{
    String temp = doubleToString( d, decPlaces );
    // if ( d  >= 0 ) temp = ' ' + temp;
    return rAlign(temp,field);
}

// Left Align gets an extra space on left to line-up with
// negative sign
public static String lAlign (double d, int field, int decPlaces)
{
    String temp = doubleToString( Math.abs(d), decPlaces );
    if ( d < 0 ) temp = '-' + temp;
    else   temp = ' ' + temp;
    return lAlign(temp,field);
}

}//end of class