CSC 175 - Intermediate Programming

Lecture #2 - Conditional Loops and Modular Programming

The Problem with Counting Loops

- Counting loops allows us to perform a statement or a block of statements a certain number of times.
- The problem is that we do not always know exactly how many times to perform the statements in a loop in every situation.

The Problem with Counting Loops (continued)

- Let's take another look at our payroll program:
  - We do not always know how payroll records that we have.
  - It isn't very convenient to have to count the records, especially if it's a big number.
  - Wouldn't it be better if we could keep going until we enter some special value to tell the computer to stop?

Conditional Loops

- Conditional loops allow us to do this.
- ***Conditional*** loops keep repeating as long as some ***condition*** is true (or until some condition ***becomes*** true).
- Steps in solving a problem that involve ***while***, ***until***, ***as long as*** indicate a conditional loop.

While Loops

- The most common form of conditional loops are ***while*** loops.
- In Java, they have the form:

```
while (condition)
   statement;
```

or

```
while(condition)  {
   statements
}
```

A simple example - ***KeepAsking***

```
import java.util.Scanner;

public class PickPositive  {
  // A simple example of how while works
  public static void main(String[] args)  {
    Scanner keyb = new Scanner(System.in);
    int number;

    //Get your first number
    System.out.println
       ("Hi there.  Pick a positive integer");
    number = keyb.nextInt();
```

```
    //Keep reading number as long as they are
    // positive
    while (number > 0) {
      System.out.println
            ("Pick another positive integer");
      number = keyb.nextInt();
    }

    System.out.println
         (number + " is not a positive integer");
  }
}
```

Sentinel Value

- Often conditional loops continue until some special value is encountered in the input which effectively tells the program to stop running the loop. This is called a ***sentinel value*** because it is the value for which we are watching.
- -1 is the sentinel value in the GPA algorithm's main loop

The *TestAverage* Program

```
import java.util.Scanner;

public class CalcGrade  {
  // Calculates the average test grade and
  // converts it to a letter grade assuming that
  // A is a 90 average, B is an 80 average and so
  // on.that

  public static void main(String[] args)  {
    Scanner keyb = new Scanner(System.in);
    final int sentinelGrade = -1;
    int thisGrade, numTests = 0, total, thisGrade;
    float testAverage;
    char  courseGrade;

    // Initially, the total is 0
    total = 0;
```

```
    // Get the first grade
    System.out.println
        ("What grade did you get on your first test  ?");
    System.out.println("Enter -1 to end");
    thisGrade = keyb.nextInt();

    //Add up the test grades
    while (thisGrade != sentinelGrade) {
      // Make sure that the grades are valid percentages
      if (thisGrade > 100)
        System.out.println
               ("This is not a valid test grade.");
      else if (thisGrade >= 0) {
        total = total + thisGrade;
        numTests;++
      else
        System.out.println
                    ("This is not a valid test grade.");

      System.out.println
              ("What grade did you get on this test  ?");
      thisGrade = keyb.nextInt();
    }
```

```
    // Find the average
    testAverage = total/numTests;

    // Find the letter grade corresponding to the average
    if (testAverage >= 90)
      courseGrade = 'A';
    else if (testAverage >= 80)
      courseGrade = 'B';
    else if (testAverage >= 70)
      courseGrade = 'C';
    else if (testAverage >= 60)
      courseGrade = 'D';
    else
      courseGrade = 'F';

    // Print the results.
    System.out.println("Your test average is "
                                        + testAverage);
    System.out.println("Your grade will be "
                                    + courseGrade);
  }
}
```

Magic Number Problem

- The magic number game involves guessing a number and with each wrong guess, the player is told “too high” or “ too low”. The goal is to guess the number in the smallest number of tries.
- We need a method for having the computer pick a number at random for the player to guess.
- We will need to learn about how to use “library functions” to provide us with the magic number.

`import` and Standard Classes (rand)

- It is frequently helpful to be able to use software routines that have already been written for common tasks.
- **`System.out.println`** and **`keyb.nextInt()`** are examples of this.
- **`import`** allows us to access entire libraries of routines that are part of one or more ***classes***
- When we write:

  ```
  import  java.util.Scanner;
  ```

  We are telling the Java compiler where to find the definitions of the Scanner class.

`import` and Standard classes (**`Random`**)

- To use the random number function, we need to include

  ```
  import        java.util.*;
  ```

- This tells the computer that java.util contains subdirectories with class definitions that it will need to use.
- A class is similar to a data type but it can be defined by a programmer or may come as a standard part of the programming language. Classes need to be initialized before use:

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

- The name of the random number function that we want is **`nextInt()`** – it is part of the object that we will declare called **`newRandomNumber`**.

The Magic Number Program

```
import java.util.*;

public class MagicNumber  {
  // The magic number game has the user trying to
  // guess which number between 1 and 100 the
  // computer has picked
  public static void main(String[] args)  {
    Scanner keyb = new Scanner(System.in);
    Random newRandomNumber = new Random();
    int magic, guess;
    int tries = 1;

    // Use the random number function to pick a
    // number
    magic = newRandomNumber.nextInt(100) + 1;
```

```
    // Let the user make a guess
    System.out.println("Guess ?");
    guess = keyb.nextInt();

    while (guess != magic) {
      // If the user won, tell him/her
      if (guess == magic) {
        System.out.println("** Right!! ** ");
        System.out.println(magic
            + " is the magic number\n");
      }

      // Otherwise tell him whether it's too high
      // or too low
      else if (guess > magic)
        System.out.println
                  (".. Wrong .. Too high\n");
```

```
      else
        System.out.println(".. Wrong .. Too low\n");
      // Let the user make another guess
      System.out.println("Guess ?");
      guess = keyb.nextInt();
      tries++;
    }
    // Tell the user how many guesses it took
    System.out.println("You took " + tries
                    + " guesses\n");
  }
}
```

Declaring Boolean Constants

- If we want to work with ***true*** and ***false*** we can work with **`boolean`** variables.
- We can write:

```
boolean      married = true;
… … … …
if (married)
 System.out.println("The employee is
   married\n");
```

`!` operator

- Sometimes we want to test to see if a condition is ***not*** true.
- We can do this by using the not operator, which is written as `!`:

```
if (!married)
   System.out.println("Do you"
       + " want to bring a"
       + " date? ");
```

`&&` and `||` Operators

- Sometimes there may be two or more conditions to consider.For this reason we have the `&&` (*AND*) and `||` (*OR*) operators.
- If we declare
- `boolean p, q;`
- …
- Both `p` and `q` must be true for `p && q` to be true.
- `p || q` is true unless both `p` and `q` are false.

do.. while loops

- You may have noticed that we asked the user twice for same information - the number (s)he is guessing.
- Some loops really require that the condition be at the end - not at the beginning.
- In Java, we have the do.. while loop, whose syntax is:

```
do      {
   statement(s)
} (condition)
```

Revisiting the magic number program

- The main loop in the magic number program becomes:

```
do{
  // Let the user make a guess
  System.out.println("Guess: ");
  guess = keyb.nextInt();
  // If the user won, tell him/her
  if (guess == magic)   {
      System.out.println("** Right!! ** ");
      System.out.println(magic
            + " is the magic number\n");
  }
```

Revisiting the magic number program (continued)

```
  // Let the user make another guess
  else if (guess > magic)
    System.out.println(".. Wrong .. Too high\n");
  else
    System.out.println(".. Wrong .. Too low\n");
  tries++;
} while (guess != magic);
```

What are methods?

- We have seen a few examples of procedures (in Java, we call them methods):
  - **`System.out.println`**, which we have used to display output on the screen
  - **`Keyb.nextInt`**, which we have used to get integer inputs from the keyboard
  - **`newRandomNumber.nextInt()`**, which we have used to get a random numbers
- Functions allow us to use software routines that have already been written (frequently by other people) in our programs.

  E.g., **`magic = newRandomNumber.nextInt();`**

What are parameters?

- A ***parameter*** is a value or a variable that is used to provide information to a function that is being called.
- If we are writing a function to calculate the square of a number, we can pass the value to be squared as a parameter:

```
printSquare(5);
printSquare(x)
```

actual parameter

- These are called actual parameters because these are the actual values (or variables) used by the function being called.

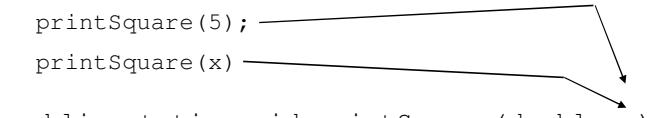
Formal Parameters

- Functions that use parameters must have them listed in the function header. These parameters are called ***formal parameters.***

```
public static void printSquare(double x)  {
   double square;
   square = x*x;
   System.out.println("The square of "
         + x + " is " + square);
 }
```

formal parameters

Parameter Passing




```
public static void printSquare(double x)
{
  double square;
  square = x*x;
  System.out.println("The square of "
        + x + " is " + square);
}
```

In both cases, calling the function requires copying the actual parameter's value where the function can use it. Initially, x has whatever value the actual parameter has.

Parameter Passing (continued)


```
public static void printSquare(double x)
{
  double square;
  square = x*x;
  System.out.println("The square of "
        + x + " is " + square);
}
```

x *initially is set to* 5. *square is then set to the value of* x^2 *or* 5^2 *or* 25.

Parameter Passing (continued)

```
printSquare(x)
```

```
public static void printSquare(double x)
{
  double square;
  square = x*x;
  System.out.println("The square of "
        + x + " is " + square);
}
```

x *initially is set to whatever value x had in the main program. If x had the value 12, square is then set to the value of* x^2 *or* 12^2 *or* 144.

The **Squares** Program

```
import java.util.Scanner;

public class Squares {
  // main() - A driver for the print_square
  //          function
  public static void main(String[] args)  {
    Scanner keyb = new Scanner(System.in);
    double  value;

    // Get a value and print its square
    System.out.println("Enter a value ?");
    value = keyb.nextDouble();
    printSquare(value);
  }
```

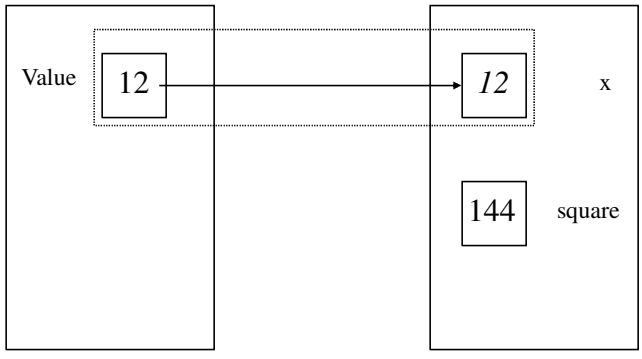
the actual parameter in the function call

the actual parameter in the function call

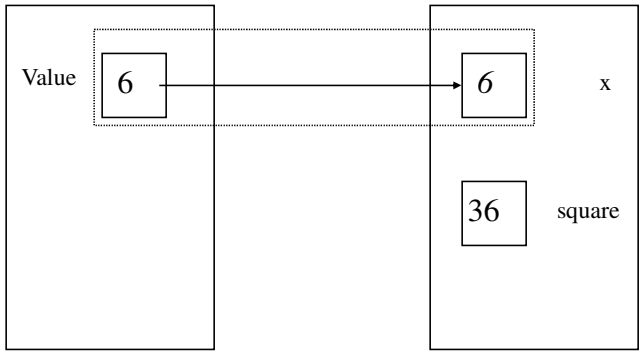
```
  // printSquare() - Prints the square of whatever
  //                 value that it is given.
  public static void print_square(double x)  {
    double  square;

    square = x*x;
    System.out.println("The square of " + x
                          + " is " + square);
  }
}
```

Passing Parameters - When The User Inputs 12



Passing Parameters - When The User Inputs 6



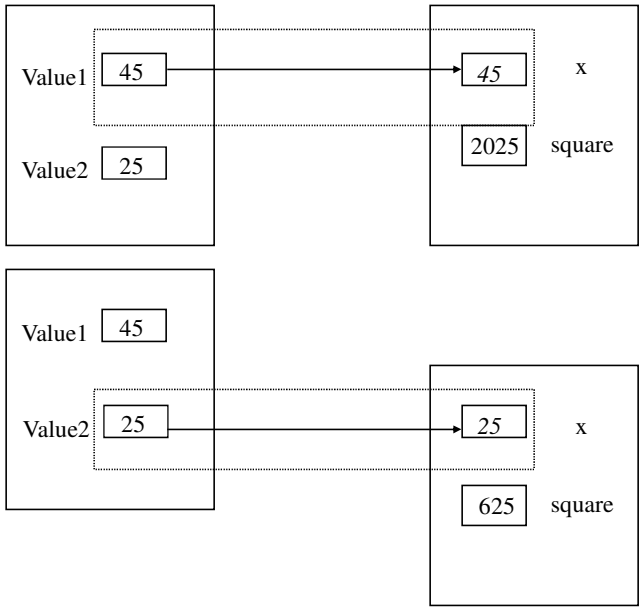
A Rewrite of **`main`**

```
import java.util.Scanner;

public class Squares2 {
  // main() - A driver for the print_square
  //          function
  public static void main(String[] args)  {
    double value1 = 45, value2 = 25;

    printSquare(value1);
    printSquare(value2);
  }
```

Passing Parameters - Using **`square`** Twice In One Program



A program to calculate Grade Point Average

Example - Ivy College uses a grading system, where the passing grades are A, B, C, and D and where F (or any other grade) is a failing grade. Assuming that all courses have equal weight and that the letter grades have the following numerical value:

Letter grade	Numerical value
A	4
B	3
C	2
D	1
F	0

write a program that will calculate a student's grade point average.

Let's Add– Dean's List

- Let's include within the program a method that will print a congratulatory message if the student makes the Dean's List.
- We will write a function **`deansList`** that will print the congratulatory message and another method **`printInstructions`**.

A program to calculate Grade Point Average

Input - The student's grades
Output - Grade point average and a congratulatory message (if appropriate)
Other information

"A" is equivalent to 4 and so on

GPA = Sum of the numerical equivalents/ Number of grades

Our first step is to write out our initial algorithm:

1. Print introductory message
2. Add up the numerical equivalents of all the grades
3. Calculate the grade point average and print it out
4. Print a congratulatory message (if appropriate)

The Entire **`DeansList`** Program

```
import java.util.Scanner;

public class DeansList  {
  // Calculates a grade point average assuming
  // that all courses have the same point value
  // and that A, B, C and D are passing grades and
  // that all other grades are failing.
  public static void main(String[] args)  {
    Scanner keyb = new Scanner(System.in);
    int numCourses = 0;
    char grade;
    String inputString = new String();
    double gpa, total = 0;

    printInstructions();
```

```
    // Get the first course grade
    System.out.println("What grade did you get in"
           " your first class?");
    inputString = keyb.next();
    grade = inputString.charAt(0);
```

```
    // Add up the numerical equivalents of
    // the grades
    while (grade != 'X') {
      //Convert an A to a 4, B to a 3, etc.
      // and add it to the total
      if (grade == 'A')
        total = total + 4;
      else if (grade == 'B')
        total = total + 3;
      else if (grade == 'C')
        total = total + 2;
      else if (grade == 'D')
        total = total + 1;
      else if (grade != 'F')
        System.out.println("A grade of " + grade
                    + " is assumed to be an F\n");
      numCourses++;
```

```
      // Get the next course grade
      System.out.println
          ("What grade did you get in the"
                           + " next class?");
      inputString = keyb.next();
      grade = inputString.charAt(0);
    }

    // Divide the point total by the number of
    // classes to get the grade point average
    // and print it.
    gpa = total / numCourses;
    System.out.printf
     ("Your grade point average is %4.2f\n", gpa);
    deansList(gpa);
  }
```

```
// printInstructions() - Prints instructions
//                       for the user
public static void printInstructions()  {
  // Print an introductory message
  System.out.println
     ("This program calculates your grade point"
          + " average\n");
  System.out.println
     ("assuming that all courses have the same"
          + "point \n");
  System.out.println
     ("value.  It also assumes that grades of "
          + "A, B, C and D\n");
  System.out.println
     ("are passing and that all other grades "
          + "are failing.\n");
  System.out.println
     ("To indicate that you are finished, "
               + "enter a grade of \'X\'\n\n");
}
```


```
// printInstructions() - Prints instructions
//                       for the user
public static void printInstructions()  {
  // Print an introductory message
  System.out.println
     ("This program calculates your grade point"
          + " average\n");
  System.out.println
     ("assuming that all courses have the same"
          + "point \n");
  System.out.println
     ("value.  It also assumes that grades of "
          + "A, B, C and D\n");
  System.out.println
     ("are passing and that all other grades "
          + "are failing.\n");
  System.out.println
     ("To indicate that you are finished, "
               + "enter a grade of \'X\'\n\n");
}
```

```
  // deansList() - Print a message if (s)he made
  //                dean's list
  public static void deansList(double gpa)  {
    if (gpa >= 3.2)
      System.out.println
        ("Congratulations!! You made"
          + " dean\'s list!!\n\n");
  }
}
```

Example – x to the nth power

- Let's write a function to calculate x to the nth power and a driver for it (a main program whose sole purpose is to test the function.
- Our basic algorithm for the function:
  - Initialize (set) the product to 1
  - As long as n is greater than 0:
    - Multiply the product by x
    - Subtract one from n

power Program

```
import java.util.Scanner;

public class Power  {
  // A program to calculate 4-cubed using a
  // function called power
  public static void main(String[] args)  {
    double x, y;
    int n;

    x = 4.0;
    n = 3;
    y = 1.0;
    power(y, x, n);
    System.out.println("The answer is " + y);
  }
```

```
  // power() -  Calculates y = x to the nth power
  public static void power(double y,
                           double x, int n)  {
    y = 1.0;
    while (n > 0) {
      y = y * x;
      n = n - 1;
    }
    System.out.println("Our result is " + y);
  }
}
```

The Output From **`power`**

Our result is 64
The answer is 1

Shouldn't these be the same numbers?

The problem is that communication using parameters has been one-way – the function being called listens to the main program , but the main program does not listen to the function.

Value Parameters

- The parameters that we have used all pass information from the main program to the function being called by copying the values of the parameters. We call this ***passing by value***, because the value itself is passed.
- Because we are using a copy of the value copied in another location, the original is unaffected.

Methods and Functions

- Some methods perform specific tasks and do not produce any one data item that seem to be their whole reason for existence.
- Other methods are all about producing some value or data item; in many programming languages they are called ***functions***.

`void` Functions

- Normally a function is expected to produce some ***result*** which is returns to the **`main`** program:

```
average = calcAverage(x, y, z);
```

- The data type of the function's result is also called the function's type.
  - Functions that produce an integer are called ***integer functions***.
  - Functions that produce float value are called ***float functions***.
  - Functions that do not produce a result are called ***void functions***
- When we write

```
public void printSquare(int x);
```

  it means that the function is not expected to return a result.

Writing Functions That Return Results

- We can write a function that returns a result by replacing that void with a data type:

```
public double average3(int a, int b, int c);
```

```
public double  average3(int a, int b, int c)  {
      float sum, mean;
      sum = a + b + c;
      mean = sum / 3;
      return mean;
}
```

The result that we are returning is mean

Writing Functions That Return Results

- The syntax is:

  `return` *expression*`;`

- Return statements have contain expressions, variables, constants or literals:

```
return true;
return 35.4;
return sum;
return sum/3;
```

Rewriting the **`average3`** Function

```
public double average3(int a, int b, int c)
{
      float sum, mean;
      sum = a + b + c;
      return sum / 3;
}
```

Maximum and Minimum

- Let's write a pair of functions that find the minimum and maximum of two values ***a*** and ***b***.
- Initial algorithm for maximum:

  Return the larger of a and b:
- If we refine this:

  1.1 IF a > b return a

  1.1 else return b //a < = b

•For minimum, we replace > with <

```
public double maximum(float x, float y)
{
     if (x > y)
          return(x);
     else
          return(y);
}
```

```
public double minimum(float x, float y)
{
     if (x < y)
          return(x);
     else
          return(y);
}
```

Rewriting the Payroll Program

```
import java.util.Scanner;

public class Payroll3  {
  // A simple payroll progam that uses a method
  // to calculate the gross pay
  public static void main(String[] args)  {
    Scanner keyb = new Scanner(System.in);
    double hours, rate, pay;

    // Ask the user for payrate
    System.out.println
      ("What is rate of pay for the employee?");
    rate = keyb.nextDouble();

    // Enter the hours worked
    System.out.println("Enter the hours worked?");
    hours = keyb.nextInt();
```

```
    // Get the gross pay
    pay = gross(hours, rate);
    System.out.printf
          ("Gross pay is $%4.2f\n",pay);
  }

  // gross() -  Calculate the gross pay.
  public static double gross(double hours,
                                double rate)  {
    double pay;

    // If hours exceed 40, pay time and a half
    if (hours > 40)
      pay = 40*rate + 1.5*rate*(hours-40);
    else
      pay = rate * hours;
    return pay ;
  }
}
```

`return`

- return serves two purposes:
  - It tells the computer the value to return as the result.
  - It tell the computer to leave thje function immediately and return the main program.

```
// gross() -  Calculate the gross pay.
 public static double gross(double hours,
                                 double rate) {
   // If hours exceed 40, pay time and a half
   if (hours > 40)
     return(40*rate + 1.5*rate*(hours-40));
   return(rate*hours);
 }
```

Rewriting pow

- We can make the pow function tell the main program about the change in **`y`** by having it return the value as the result:

```
public static double power(double x,
                                      int n) {
     … …
}
```

The rewritten **pow** program

```
import java.util.Scanner;

public class PowerTest  {
  // A program to calculate 4-cubed using a
  // function called power
  public static void main(String[] args)  {
    double x, y;
    int n;

    x = 4.0;
    n = 3;
    y = power(x, n);
    System.out.println("The answer is " + y);
  }
```

```
  // power() -  Calculates y = x to the nth
  //            power
  public static double power(double x, int n) {
    double prod;

    prod = 1.0;
    while (n > 0) {
      prod = prod * x;
      n = n - 1;
    }
    System.out.println("Our result is "
                                    + prod);
    return prod;
  }
}
```

The New Output From **power**

Our result is 64
The answer is 64

Exactly what we would expect Why?

Communication using the result is two-way – the function being called listens to the main program, but the main program listens to the function because data changes are explicitly passed back to the main method.

An Example – **square2**

- Let's rewrite the square program so that the function calculates the square and passes its value back to the main program, which will print the result:

```
import java.util.Scanner;

public class Square2  {
  // This illustrates how to use methods to
  // find the square of a value

  // main() - A driver for the findSquare method
  public static void main(String[] args)  {
    Scanner keyb = new Scanner(System.in);
    double  value, square;

    System.out.println("Enter a value ?");
    value = keyb.nextDouble();
```

```
    square = findSquare(value);
    System.out.println("The square of " + value
                        + " is " +  square);
  }

  // findSquare() - Calculates the square of
  //                whatever value it is given.
  public static double findSquare(double x)  {
    double square = x*x;
    return square;
  }
}
```

Comparing print_square and find_square

- What are the differences between **`print_square`** and **`find_square`**?
- **`print_square`**:
  - uses value parameters
  - prints the square; it doesn't have to pass that value to the main program
- **`find_square`**:
  - Returns the result
  - does not print the square; it must pass the value back to the main program

Example: Average3

- Let's write a program which will find the average of three numbers:
- Our algorithm is:

1. Read the values
2. Calculate the average
3. Print the average

Average3c.java

```
import java.util.Scanner;
public class Average3c  {
  // Find the average of three numbers using a
  // function
  public static void main(String[] args)  {
    int value1, value2, value3;
    double average;

    //Get the inputs
    value1 = getValue();
    value2 = getValue();
    value3 = getValue();

    // Call the function that calculates the
    // average
    average = findAverage(value1, value2, value3);
    System.out.println
                ("The average is " + average);
  }
```

```
  // getValue() - Prompt the user and read a value
  public static int getValue()  {
    Scanner keyb = new Scanner(System.in);
    System.out.println("Enter a value ?");
    int x = keyb.nextInt();
    return x;
  }

  // find_average() -  Find the average of three
  //                   numbers
  public static double findAverage(int x, int y,
                                           int z)  {
    double sum = x + y + z;
    double average = sum / 3;
    return average;
  }
}
```

Preconditions and Postconditions

- Preconditions – are conditions that we expect and require to be true ***before entering*** the procedure
- Postconditions– are conditions that we expect and require to be true ***after exiting*** the procedure
- Examples in square3:
  - getinput has a **postcondition** that a value was read in and that the value is set.
  - find average has a **precondition** that all value1, value2 and value al have values.