Loops – While, Do, For

- Repetition Statements
  - While
  - Do
  - For
- Introduction to Arrays
- Reading for this Lecture, L&L, 5.4,6.3-6.4, 8.1-8.2

Repetition Statements

- *Repetition statements* allow us to execute a statement or a block of statements multiple times
- Often they are referred to as *loops*
- Like conditional statements, they are controlled by boolean expressions
- Java has three kinds of repetition statements:

  while
  do
  for

- The programmer should choose the right kind of loop statement for the situation

The while Statement

- A *while statement* has the following syntax:

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

- If the `condition` is true, the `statement` is executed
- Then the condition is evaluated again, and if it is still true, the statement is executed again
- The statement is executed repeatedly until the condition becomes false

The while Statement

- An example of a while statement:

```
boolean done = false;
while (!done)
{
   body of loop statements;
   if (some condition)
      done = true;
}
```

- If the condition of a `while` loop is false initially, the statement is never executed
- Therefore, the body of a `while` loop will execute zero or more times

The while Statement

- Let's look at some examples of loop processing
- A loop can be used to maintain a *running sum*
- A *sentinel value* is a special input value that represents the end of input (not valid as data!)
- See Average.java (page 237)
- A loop can also be used for *input validation*, making a program more *robust*
- See WinPercentage.java (page 239)

Infinite Loops

- Executing the statements in the body of a `while` loop must eventually make the condition false
- If not, it is called an *infinite loop*, which will execute until the user interrupts the program
- This is a common logical error
- You should always double check the logic of a program to ensure that your loops will terminate

Infinite Loops

- An example of an infinite loop:

```
boolean done = false;
while (!done)
{
   System.out.println (“Whiling away the time ...”);
   // Note: no update for the value of done!!
}
```

- This loop will continue executing until the user externally interrupts the program

Nested Loops

- Similar to nested `if` statements, loops can be nested as well
- That is, the body of a loop can contain another loop
- For each iteration of the outer loop, the inner loop iterates completely
- See PalindromeTester.java (page 243)

Nested Loops

- How many times will the string `"Here"` be printed?

```
count1 = 1;
while (count1 <= 10)
{
   count2 = 1;
   while (count2 <= 20)
   {
      System.out.println ("Here");
      count2++;
   }
   count1++;
}
```

10 * 20 = 200

The do Statement

- A *do statement* has the following syntax:

```
do
{
   statement;
}
while ( condition );
```

- The `statement` is executed once initially, and then the `condition` is evaluated

- The statement is executed repeatedly until the condition becomes false

The do Statement

- An example of a do loop:

```
boolean done = false;
do
{
    body of loop statements;
    if (some condition)
        done = true;
} while (!done);
```

- The body of a do loop executes one or more times (Note: At least once!)
- See ReverseNumber.java (page 252)

The for Statement

- A *for statement* has the following syntax:

The ***initialization*** is executed once before the loop begins

The ***statement*** is executed until the ***condition*** becomes false

```
for ( initialization ; condition ; increment )
   statement;
```

The ***increment*** portion is executed at the end of each iteration

The for Statement

- A `for` loop is functionally equivalent to the following `while` loop structure:

```
initialization;
while ( condition )
{
   statement;
   increment;
}
```

The for Statement

- An example of a `for` loop:

```
for (int count=1; count <= 5; count++)
   System.out.println (count);
```

- The initialization section can be used to declare an int variable for counting
- Like a `while` loop, the condition of a `for` loop is tested prior to executing the loop
- Therefore, the body of a `for` loop will execute zero or more times

The for Statement

- The increment section can perform any calculation

```
for (int num=100; num > 0; num -= 5)
   System.out.println (num);
```

- A `for` loop is well suited for executing the body a specific number of times that can be calculated or determined in advance
- See Multiples.java (page 256)
- See Stars.java (page 258)

The for Statement

- Each expression in a `for` statement is optional
- If the initialization is left out, no initialization is performed
- If the condition is left out, it is always considered to be true, and therefore creates an infinite loop
- If the increment is left out, no increment operation is performed
- “Loop forever” can be written as:

```
for (;;)
    {body;}
```

Introduction to Arrays

- It is very useful to have a group of variables that can be processed in a loop where one variable is processed during each pass through the loop
- But we don't want to declare them as individual variables, e.g. five individual integer variables:

```
int num0, num1, num2, num3, num4;
```

- We can't use a loop index variable to refer to one variable num0, num1, etc without a lot of nested if-else statements or a switch statement

Introduction to Arrays

- Without arrays we would need to do something like this (NOTE: Don't do it this way!):

```
int num0, num1, num2, num3, num4;
for (int i = 0; i < 5; i++) {
   switch (i) {
        case 0:
             statements using num0;
             break;
        case 1:
             same statements using num1;
             break;
        // three more cases needed here
   }
}
```

Introduction to Arrays

- We can declare an array of variables of a specific type with the capability to use an index variable to select one variable

```
int [ ] nums = new int [5];
```

- The above declares 5 variables of type int
- The valid array index values are `0-4` (not `1-5`)
- Note: Values have not been assigned to those 5 variables in the array yet.

Introduction to Arrays

- To assign values to each variable, we can use a for-loop:

```
for (int i = 0; i < 5; i++)
  nums[i] = some valid integer expression;
```

- A single int variable can be selected using an integer expression or value inside the [ ]:

```
int result = nums[integer expression];
```

Arrays and Initializer Lists

- An array can be defined and initialized so that each element contains a specific value:

```
char [] vowels = {‘a’, ‘e’, ‘i’, ‘o’,
  ‘u’};
```

- Java uses the initializer list to determine how long the array must be and allocates that many elements
- An initializer list can be used only when the array is first declared, as above
- Afterward, each individual element of the array

Arrays and Loops

- Now we can coordinate the processing of one variable with the execution of one pass through a loop using an index variable, e.g:

```
int MAX = 5; // symbolic constant
int [ ] nums = new int [MAX];
for (int i = 0; i < MAX; i++) {
  // use i as array index variable
  Java statements using nums[i];
}
```

Alternative Loop Control Condition

- Arrays are objects (but, not based on a class)
- Each array has an *attribute* “length” that we can access to get a value equal to the length of that array, e.g.nums.length is equal to MAX:

```
int MAX = 5; // symbolic constant
int [ ] nums = new int [MAX];
for (i = 0; i < nums.length; i++) {
   // use i as array index variable
   in Java statements using nums[i];
}
```

Method versus Attribute

- Remember that the String class had a length **method**, that we accessed as:

```
int length = stringName.length();
```

- For an array length, we access a length **attribute** not a method so there is no ():

```
int length = arrayName.length;
```

- We will get into this subtle distinction in more detail after the first exam.