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* – better known as ***loops*** – allow us to execute code multiple times
- The repetition is controlled by boolean expressions
- Java has three kinds of loops:

  while
  do-while
  for

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

The while Loop

- A *while loop* has the following syntax:

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

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

The while Loop

- An example of a while loop:

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

- If the condition of a `while` loop is false to begin with, the statement is **never** executed
- Therefore, the body of a `while` loop will execute **0+ times**

The while Loop

- Let's look at some examples of loop processing
- A loop can be used to maintain a *running sum* (for example, our dice games!)
- You can have a flag or signal (called a *sentinel value*) that represents the end of input (not data!) and stops the loop
- A loop can also be used for *input validation*, making a program more *robust*

Infinite Loops

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

- Executing ***statement*** must eventually make ***condition*** false
- If not, you have an *infinite loop*, which will run until the user interrupts the program
- This is a common **logical** error
- You should always double check the logic of your program to ensure that your loops will eventually **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 go on forever (*in theory, at least!*) until the user externally interrupts the program

Nested Loops

- As with `if` statements, you can have loops inside of loops!
- For each iteration of the outer loop, the inner loop runs through completely
- See PalindromeTester.java

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-while Loop

- A *do-while loop* looks like this:

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

- **statement** is executed once initially, guaranteed to run at least once, and then the **condition** is evaluated
- **statement** is executed repeatedly until **condition** becomes false

The do-while Loop

- An example of a `do-while` loop:

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

- The body of a `do` loop executes **1+ times** (versus the **0+** times of `while`)
- See ReverseNumber.java

The for Loop

- A *for loop* 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 Loop

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

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

The for Loop

- 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 **0+ times**

The for Loop

- 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
- See Stars.java

The for Loop

- 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 whole group of variables that can be processed sequentially in a loop
- But we don't want to declare them as individual variables like this:

```
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: Please **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 a whole group (called an **array**) of variables of a specific type

```
int[] nums = new int [5];
char[] chars = new char[10];
```

- You can have arrays of objects, as well

```
String[] strings = new String[5];
```

- Note: Those variables in the arrays have not been initialized yet.

Introduction to Arrays

- You access a variable within an array by it's **index**.
- Indices start at 0 (not 1). To illustrate, take an array at location X using a type of size `typeSize`:

- 1st is at X, the 2nd at X + (`typeSize`), the third at X + (2*),..., the Nth at X + ((N-1) * `typeSize`).
- Given an array `items`, the first one is `items[0]`, the fifth one `items[4]`, etc.

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 variable can be selected using an integer expression or value inside the [ ]:

```
count = 8;

int result = nums[count];
int otherResult = nums[count * 3 % 5];
```

Arrays and Initializer Lists

- An array can be defined and initialized with an an initializer list (an array literal):

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

- Java allocates right amount of space based upon the list size
- An initializer list can be used only when the array is first declared, as above
- Afterward, each element of the array can be accessed with an index, per usual:

```
boolean result = vowels[3] == 'o'   // true
```

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];
}
```

Arrays and Loops

- Arrays are objects (only without a class)
- Each array has an *attribute* "length" that we can access to get the length of that array, e.g., nums.length == MAX:

```
int MAX = 5; // symbolic constant
int [ ] nums = new int [MAX];
for (int 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 array length, we use a length **attribute** not a method, hence no ()

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

- The distinction is subtle but important, and we will get into it in more detail after the first exam.