Outline

- Problem: How can I perform the same operations a fixed number of times?

- Considering “for loops”
  - Performs same operations as while and do-while

- Structure provides more compact representation

- Arrays
  - Efficient representation of large amount of data
The for loop

- A for statement has the following syntax:

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

- The initialization is executed once before the loop begins.
- The statement is executed until the condition becomes false.
- The increment portion is executed at the end of each iteration.

Logic of a for loop

Does initializing, pretest, increment and posttest …..
A for loop is functionally equivalent to the following while loop structure:

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

Know how to write the same functionality in ALL loops.

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 a variable

Like a while loop, the condition of a for loop is tested prior to executing the loop body

Therefore, the body of a for loop will execute zero or more times
The for Statement

- The increment section can perform any calculation

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

- A for loop is well suited for executing statements a specific number of times

- Question: Can I use `num` outside of the loop?

The for Statement

- ➡ Each expression in the header of a for loop 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

- continue keyword
- break keyword
**Break & Continue**

° Continue statement means “skip to the end of the loop and do the next iteration”

° Break means “skip to the end of the loop and exit the loop”.

```java
for(int idx = 0; idx < 10; idx++)
{
    if(idx <  4)
        continue;
    if(idx > 7)
        break;
    System.out.println("Counter is " + idx);
}
```

**Example of break**

```java
int sum = 0;
int item = 0;

while (item < 5)
{
    item ++;
    sum += item;
    if (sum >= 6) break;
}
System.out.println("The sum is " + sum);
```

**Break makes it difficult to determine how many times the loop is executed**
Example of `continue`

```java
int sum = 0;
int item = 0;
while (item < 5) {
    item++;
    if (item == 2)
        continue;
    sum += item;
}
System.out.println("The sum is " + sum);`Continue can be used for special conditions`
For Loop Advice

- Try to limit nesting to no more than three levels of loops in one section of code.
- Try to keep the bodies of loops fairly small, ideally within one page view.
- Be even more careful about using breaks or continues in very long loop bodies.

```java
for (int num=100; num > 0; num -= 5)
{
    if (num == 10);
    break;
}
```

---

From *while* to *for*

```java
int i = startValue;
while (i < endValue)
{
    ....
    i++;
}
```

```java
for (int i=startValue; i<endValue; i++)
{
    ...
}
```
Nesting *for*-loops

- Inside the loop body of a *for*-loop, we can put another *for*-loop.
- Each time through the 1st *for*-loop, we execute the 2nd loop until its guard is false.
- Handy for printing tables like this:
  
  1 1 1 1 1
  2 2 2 2 2
  3 3 3 3 3
  4 4 4 4 4

Simple example

```java
for (int i=0; i<5; i++)
{
    for (int j=0; j<3; j++)
    {
        System.out.print(i+" ");
    }
    System.out.println();
}
```

How many values are printed out by these loops?
Arrays

- An array is an ordered list of values

<table>
<thead>
<tr>
<th>The entire array has a single name</th>
<th>Each value has a numeric index</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>scores</td>
<td>79</td>
</tr>
</tbody>
</table>

An array of size N is indexed from zero to N-1

This array holds 10 values that are indexed from 0 to 9

Arrays

- A particular value in an array is referenced using the array name followed by the index in brackets

- For example, the expression

```
scores[2]
```

```
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| scores | 79 | 87 | 94 | 82 | 67 | 98 | 87 | 81 | 74 | 91 |
```

refers to the value 94 (the 3rd value in the array)

- Expression represents a place to store a single integer
  - Can be used wherever an integer variable can be used
Arrays

° For example, an array element can be assigned a value, printed, or used in a calculation just like any other variable. BUT the array name and the specific entry or item in the array must be unambiguously used in the expression:

\[
\text{scores}[2] = 89;
\]
\[
a = \text{scores}[10];
\]
\[
\text{scores[first]} = \text{scores[first]} + 2;
\]
\[
\text{mean} = (\text{scores}[0] + \text{scores}[1])/2;
\]
\[
\text{System.out.println ("Top = " + \text{scores}[5]);}
\]

Need both array name and index together!

Arrays

° The values held in an array are called array elements

° An array stores multiple values of the same type – the element type

° ➤ The element type can be a primitive type or an object reference

° We can create an array of integers, an array of characters, an array of String objects, etc

° ➤ In Java, the array itself is an object that must be instantiated (will show ahead)
Arrays

- Another way to depict the scores array:

  This is telling you that scores (itself) is a Reference!
  ‘Scores’ points to the array.

Declaring Arrays

- The scores array could be declared as follows:

  ```java
  int[] scores = new int[10];
  ```

- Note the syntax, the ‘reference’ and the new object! It also says that there will be 10 scores referenced.

- The type of the variable scores is int[] (“an array of integers” or “an array of ints.”)

- Note that the array type does not specify its size, but each object of that type has a specific size

- The reference variable scores is set to a new array object that can hold 10 integers
Declaring Arrays

- Some other examples of array declarations:
  ```java
  float[] prices = new float[500];
  boolean[] flags;
  flags = new boolean[20];
  char[] codes = new char[1750];
  ```

Summary

- For loop generally requires an index
  - Indicates number of times loop will be executed

- For loop contains an initializer, condition, and condition modifier

- Arrays represent data is a series of memory locations
  - All data has same array name
  - Specific locations located with subscripts

- For loops and arrays go together well
  - Fixed number of loop iterations and array size