The if Statement

- An example of an if statement:

```java
if (sum > MAX)
    delta = sum - MAX;
System.out.println ("The sum is " + sum);
```

- First the condition is evaluated -- the value of `sum` is either greater than the value of `MAX`, or it is not.

- If the condition is true, the assignment statement is executed -- if it isn’t, it is skipped.

- Either way, the call to `println` is executed next.
Block Statements

- Several statements can be grouped together into a *block statement* delimited by braces.
- A block statement can be used wherever a statement is called for in the Java syntax rules.

```java
if (total > MAX) {
    System.out.println("Error!!");
    errorCount++;
}
```
Block Statements

° In an if–else statement, the if portion, or the else portion, or both, could be block statements

```java
if (total > MAX) {
    System.out.println("Error!!");
    errorCount++;
} else {
    System.out.println("Total: "+total);
    current = total*2;
}
```
Nested if Statements

- The statement executed as a result of an if statement could be another if statement.
- These are called nested if statements.
- An else clause is matched to the last unmatched if (no matter what the indentation implies).
- Braces can be used to specify the if statement to which an else clause belongs.

```c
if (num1 < num2)
    if (num1 < num3)
        min = num1;
    else
        min = num3;
else
    if (num2 < num3)
        min = num2;
    else
        min = num3;
```
The switch Statement

° The general syntax of a switch statement is:

```
switch (expression)
{
    case value1 :
        statement-list1
    case value2 :
        statement-list2
    case value3 :
        statement-list3
    case ...
}
```

If `expression` matches `value2`, control jumps to here.
An example of a switch statement:

```java
switch (option) {
    case 'A':
        aCount++; break;
    case 'B':
        bCount++; break;
    case 'C':
        cCount++; break;
}
```

Assumes we have a character variable named option that has a character value… as in `char option = 'C';`
Comparing Strings

- Remember that in Java a character string is an object.
- The `equals` method can be called with strings to determine if two strings contain exactly the same characters in the same order.
- The `equals` method returns a boolean result.

Assuming `name1` and `name2` are objects of type `String` and already have values, we can say:

```java
if (name1.equals(name2))
    System.out.println("Same name");
```

Note the syntax!!! This is how you do it.
Comparing Objects

° The $\texttt{==}$ operator can be applied to objects
  • It returns true if the two references are aliases of each other

° The $\texttt{equals}$ method is defined for all objects
  • It performs the same operation as the $\texttt{==}$ operator

° It has been redefined in the $\texttt{String}$ class to compare the characters in the two strings

° When you write a class, you can redefine the $\texttt{equals}$ method
  • Specify conditions which result in true
The while Statement

- An example of a while statement:

```java
int count = 1;
while (count <= 5) {
    System.out.println (count);
    count++;
}
```

- 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
Infinite Loops

- An example of an infinite loop:

```java
int count = 1;
while (count <= 25)
{
    System.out.println (count);
    count = count - 1;
}
```

- This loop will continue executing until interrupted (Control-C) or until an underflow error occurs.
The do Statement

- An example of a do loop:

```java
int count = 0;
do
{
    count++;
    System.out.println (count);
} while (count < 5);
```

- The body of a do loop executes at least once
The for loop

- A *for statement* has the following syntax:

```
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.
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 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
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
From **while** to **for**

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

```cpp
for (int i=startValue; i<endValue; i++) {
    ...
}
```
Arrays

° Another way to depict the scores array:

This is telling you that scores (itself) is a Reference!

‘Scores’ points to the array.
Why Arrays?

- Arrays give direct access to any element — no need to scan the array.

Before (no arrays):

```java
if (k == 0)
    value = score0;
else if (k == 1)
    value = score1;
else … // etc.
```

After (with arrays):

```java
value = scores[k];
```
Arrays as Parameters

- An entire array can be passed as a parameter to a method
- The reference to the array is passed,
  - Makes the formal and actual parameters aliases of each other
- Changing an array element within the method changes the original
- An individual array element can be passed to a method as well,
  - The type of the formal parameter is the same as the element type.
  - In this case, the call is ‘by value.’
Arrays of Objects

- After some `String` objects are created and stored in the array:
Arrays of Objects

- Keep in mind that String objects can be created using literals
- Following declaration creates an array object called verbs
  - Fills it with four String objects created using string literals

```java
String[] verbs = {"play", "work", "eat", "sleep"};
These are referenced by verbs[0] through verbs[3].
```
Two-Dimensional Arrays

- A *one-dimensional array* stores a list of elements.
- A *two-dimensional array* can be thought of as a table of elements, with rows and columns.
Two-Dimensional Arrays

° ➤ To be precise, in Java a two-dimensional array is an array of arrays

° A two-dimensional array is declared by specifying the size of each dimension separately:

    int[][] scores = new int[12][50];

You may interpret this as an array of 12 rows and 50 columns (short, fat...)

° A array element is referenced using two index values:

    value = scores[3][6]

° ➤ The array stored in one row can be specified using one index (scores[3] is a 1d array.)
Two-Dimensional Arrays

<table>
<thead>
<tr>
<th>Expression</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table</td>
<td>int[][]</td>
<td>2D array of integers, or array of integer arrays</td>
</tr>
<tr>
<td>table[5]</td>
<td>int[]</td>
<td>array of integers</td>
</tr>
<tr>
<td>table[5][12]</td>
<td>int</td>
<td>integer</td>
</tr>
</tbody>
</table>

This can be confusing. Look for the location of the brackets.

2D array is really an array of 1D arrays.