Looping in Java

Boring repetitive work: just what the computer ordered.

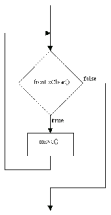
Looping in Java

- while, boolean expressions and numbers
- Printing a table of temperatures
- Spotting the structure
- The for loop: a counting loop.
- do while loops (less common)
- Example: reading input

The while statement

The **`while`** statement can be represented by a flowchart

```
while(frontIsClear())
   move();
```



The while statement

Syntax

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

Example

```
while(frontIsClear())
   move();
```

Purpose To execute a statement while a condition is true.

Looping with Numbers

- The **`while`** statement can be used with any boolean expression (as with the **`if`** statement)

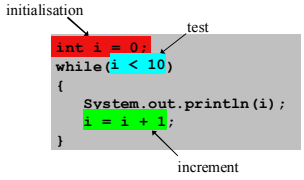
```
int i = 0;
while(i < 10)
{
   System.out.println(i);
   i = i + 1;
}
```



```
File  Edit  View  Terminal  Tabs  Help
charlie@charlie-laptop:~/ca165$ java WhileTest
0
1
2
3
4
5
6
7
8
9
charlie@charlie-laptop:~/ca165$
```

The Structure

Note the structure




```
// Print a table of squares
public class Squares
{
   public static void main(String[] args)
   {
      int i;
      i = 1;

      while (i <= 10)
      {
         System.out.print(i);
         System.out.print(" squared is ");
         System.out.println(i * i);
         i = i + 1;
      }
   }
}
```

```
// Print a table of temperatures
public class Temperature
{
   public static void main (String [] args)
   {
      final double LOW = -10.0, HIGH = 10.0;
      double cent, fahr;
      System.out.println("DEGREES C\tDEGREES F");
      cent = LOW;

      while (cent <= HIGH)
      {
         fahr = 9.0/5.0 * cent + 32.0;
         System.out.println(cent + "\t\t" + fahr);
         cent = cent + 1.0;
      }
   }
}
```

Resulting Output

DEGREES C	DEGREES F
-10.0	14.0
-9.0	15.8
-8.0	17.6
...	

- The \t is called an *escape sequence*; it represents the tab character, which helps to line up the output in columns.
- The entire block { ... } is executed each time, before the condition is checked again.

Another Example

- Print out the following song:

10 in a bed and the little one said,
"Roll over, roll over."
They all rolled over and one fell out,
9 in a bed and the little one said,
"Roll over, roll over."
They all rolled over and one fell out,
8 in a bed and the little one said,
...
1 in a bed and the little one said,
"Alone at last."

How Do We Divide It Up for Iteration?

- One possible structure:

10 in a bed and the little one said, — Before Loop
"Roll over, roll over."
They all rolled over and one fell out, — Body of Loop
9 in a bed and the little one said,
"Roll over, roll over."
They all rolled over and one fell out, — Body of Loop
8 in a bed and the little one said,
...
1 in a bed and the little one said, — Body of Loop
"Alone at last."
After Loop

Program is Organized as Follows

```
print first line of verse
while(more verses)
{
   print rest of verse
   print first line of next verse
}
print rest of last verse
```

10 in a bed and the little one said,

"Roll over, roll over."
They all rolled over and one fell out,

??? in a bed and the little one said,

"Alone at last."


```
// Print the nursery rhyme "Ten In a Bed."
public class TenInABed
{
   public static void main (String [] args)
   {
      System.out.println("10 in a bed and the little one said,");

      int numInBed = 9;
      while (numInBed > 0)
      {
         System.out.println("\t\"Roll over, roll over.\"");
         System.out.println("They all rolled over one fell out");
         System.out.println(numInBed
                  + " in a bed and the little one said,");
         numInBed = numInBed - 1;
      }
      System.out.println("\t\"Alone at last.\"");
   }
}
```

escape codes

Example: Count Number of Digits in an int

```
numberOfDigits = 0;
remainder = number;
while(remainder > 0)
{
   remainder = remainder / 10;
   numberOfDigits++;
}
```

- What happens when number < 0?
- How do we fix it?
- What about when number is exactly zero?

a common loop

This is a very common kind of loop; the variable is incremented each time in the loop

```
i = 1;
while(i <= 10)
{
   System.out.println(i);
   i = i + 1;
}
```

So Java provides a special loop form that makes it easy to do: the *for* statement

initialize, condition, increment

This while loop has three parts

initialize

```
i = 1;
while(i <= 10)
{
   System.out.println(i);
   i = i + 1;
}
```

condition

increment

So Java provides a special loop form that makes it easy to do: the *for* statement

The for loop

initialise

```
for (statement1; condition; statement2)
   statement3;
```

condition

increment

This is exactly equivalent to:

```
statement1;
while(condition)
{
   statement3;
   statement2;
}
```

Rewriting Our Square Loop

```
for(i = 1; i <= 10; i = i + 1)
{
   System.out.println(i);
   System.out.println(" squared is ");
   System.out.println(i * i);
}
```


One Advantage

- It keeps all the loop controlling statements in one place, not spread throughout the loop
- For example, if we wanted to print the table in reverse order, all changes could be made at the top of the loop:

```
for(i = 10; i > 0; i = i - 1)
{
   System.out.println(i + " squared is " + i * i);
}
```


do-while Loops

- Another form of loop in Java:

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

The body of the loop is always executed at least once; the condition is checked after the body of the loop. This is basically a convenience.

In Other Words

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

- This is exactly equivalent to writing the following while statement:

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


Example: Reading Input in a Loop

```
Enter score (-1 ends the data):  85
Enter score (-1 ends the data):  62
Enter score (-1 ends the data):  93
Enter score (-1 ends the data):  87
Enter score (-1 ends the data):  51
Enter score (-1 ends the data):  -1
5 scores were entered.
The average score is 75.6
```


Use a while Loop

```
read score
while(score != -1)
{
    process score
    read score
}
```

Short, clear, natural

Use a while Loop

```
public class Scores
{
   public static void main(String [] args)
   {
      double count = 0, total = 0;
      print("Enter score (-1 ends the data): ");
      int score = Console.readInt();
      while(score != -1)
      {
         count++;
         total = total + score;
      }
      println("There were " + count + " scores");
      println("Average is " + total / count);
   }
}
```


What the Processing Looks Like

- We'll have statements like:

```
count++;          // how many scores
total = total + score;  //update sum
```

- But for these variables count and total to be updated correctly *within* the loop, they need to be initialized correctly *before* the loop, to the value 0

So far, what do we have?

```
int count = 0;
int total = 0;

print("Enter score (-1 ends the data): ");
int score = Console.readInt( );

while(score != -1)
{
   count++;       //new score
   total = total + score; //update sum
   // count is the number of scores read
   // so far and total is their sum
   print("Enter score (-1 ends the data): ");
   score = Console.readInt( );
}
```


Afterwards

- Once the loop has ended, we can compute the average score simply as follows:

```
(double) total / count
```

- Why do we need to cast total to a double? What do we need to check before we do this computation?

Summary: Looping in Java

- while, boolean expressions and numbers
- Printing a table of temperatures
- Spotting the structure
- The for loop: a counting loop.
- do while loops (rare)
- Example: reading input

Questions?