Iteration

Software Development 1 (F27SA)

Michael Lones

Week 3, lecture 2

Today’s Lecture(s)

- What is iteration?
- `while` and `do…while` statements
- `for` statements

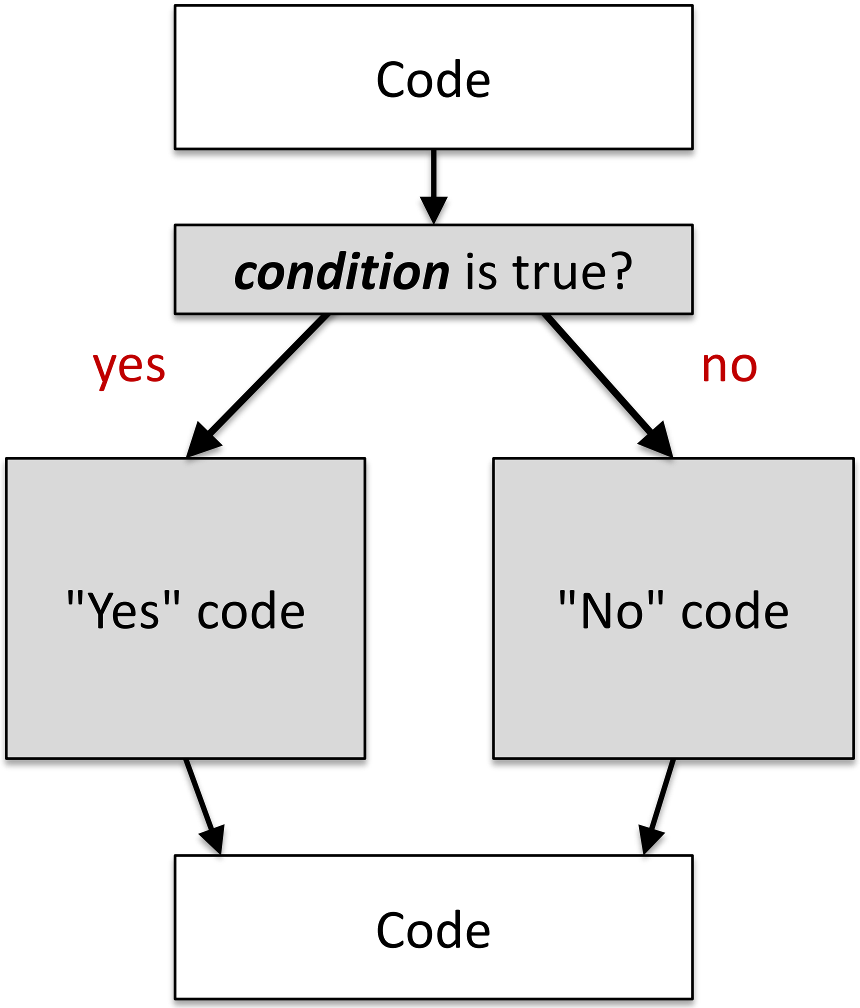
Part I
Simple Loops

What is Iteration?

Iteration is when the same group of statements are executed repeatedly within a **loop**

- For a start, this saves you from having to write the same statements over and over again
- Also, the code typically does something slightly different each time around the loop
- Iteration and conditional execution are referred to as **control flow statements**, since they determine the flow of execution inside a program

Control Flow



```
// code

if(condition) {
   // "Yes" code
}
else {
   // "No" code
}

// code
```

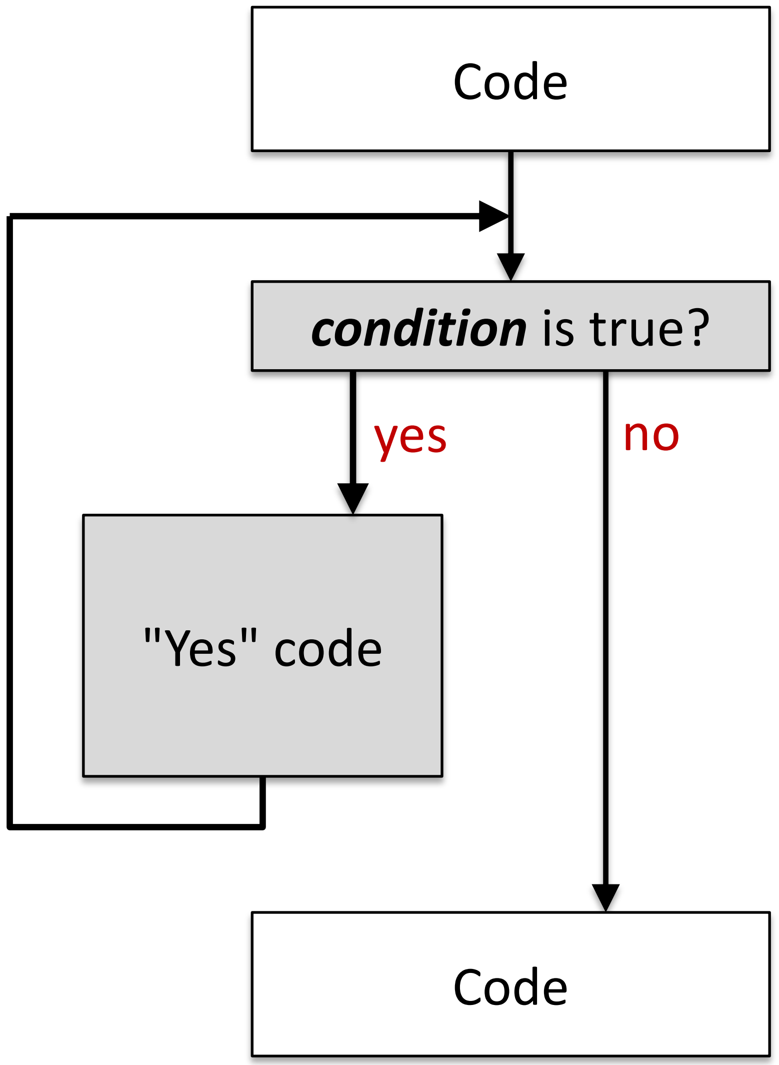
`while` loops

This repeatedly executes the statements in its *body* while a condition remains true

```
while(condition) {
    // loop body
    // comprising any number
    // of statements
}
```

The condition gets evaluated when the while statement is first encountered, and is then re-evaluated at the start of each iteration.

Control Flow



```
// code


while(condition)
{
   // "Yes" code
}


// code
```

Simple examples

WhileDemo.java

```java
public class WhileDemo {
    public static void main(String[] args) {
        String message = "Hello!";
        while(true) {
            System.out.println(message);
        }
    }
}
```

Terminal

```
$ java WhileDemo
Hello!
Hello!
Hello!
Hello!
Hello!
```

Simple examples

WhileDemo.java

```
public class WhileDemo {
   public static void main(String[] args) {
      String message = "Hello!";
      while(true) {
         System.out.println(message);
      }
   }
}
```

Terminal

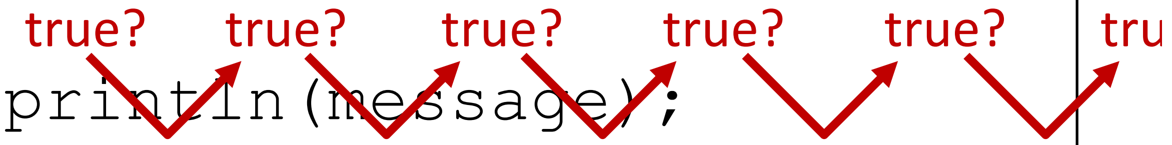
```
$ java WhileDemo
Hello!
Hello!
Hello!
Hello!
Hello!
```

Because the condition is always "true", this loop will never end. This is known as an **infinite loop**.

Simple examples

WhileDemo.java

```java
public class WhileDemo {
   public static void main(String[] args) {
      String message = "Hello!";
      while(true) {
         System.out.println(message);
      }
   }
}
```



Terminal

```
$ java WhileDemo
Hello!
Hello!
Hello!
Hello!
Hello!
```

Simple examples

Without loops, we'd have to write:

```
public class NotWhileDemo {
   public static void main(String[] args) {
      String message = "Hello!";
      System.out.println(message);
      System.out.println(message);
      System.out.println(message);
      System.out.println(message);
      System.out.println(message);
      System.out.println(message);
      … // keep typing forever!
      }
   }
}
```

Simple examples

This loop stops when the user enters "stop"

WhileDemo2.java

```
public class WhileDemo2 {
    public static void main(String[] args) {
        String input = "";
        Scanner scan = new Scanner(System.in);

        while(!input.equals("stop")) {
            System.out.println("Type stop to stop");
            input = scan.nextLine();
        }
        System.out.println("Stopped!");
    }
}
```

Simple examples

Without loops, we'd have to write:

```
public class NotWhileDemo2 {
    public static void main(String[] args) {
        String input;
        Scanner scan = new Scanner(System.in);

        System.out.println("Type stop to stop");
        input = scan.nextLine();
        if(!input.equals("stop")) {
            System.out.println("Type stop to stop");
            input = scan.nextLine();
        }
        else if(!input.equals("stop")) {
            System.out.println("Type stop to stop");
            input = scan.nextLine();
        }
        … // keep typing forever!
        System.out.println("Stopped!");
    }
}
```

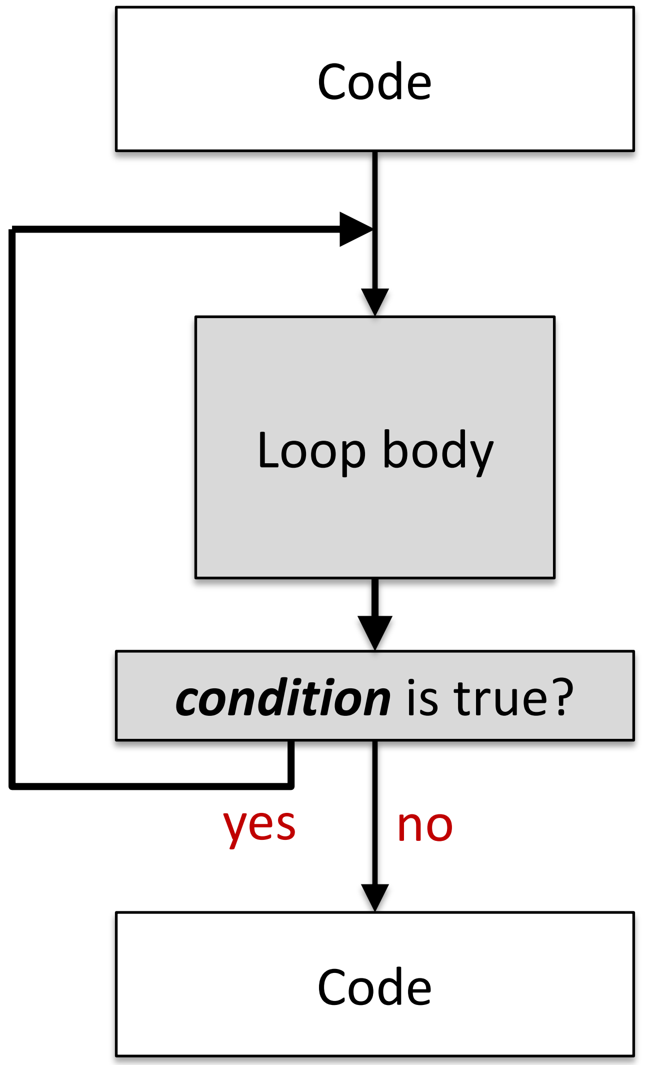
`do…while` loops

This is almost the same as a while loop, except the condition is tested at the end of each iteration

```
do {
    // loop body
    // comprising any number
    // of statements
} while(condition)
```

Which means the loop body is always executed at least once.

Control Flow



```
// code


do {
   // loop body
}
while(condition)


// code
```

Simple examples

This loop stops when the user enters a valid number

DoWhileDemo.java

```
public class DoWhileDemo {
    public static void main(String[] args) {
        int input;
        Scanner scan = new Scanner(System.in);
        do {
            System.out.println("Please enter a
                number between 1 and 5");
            input = scan.nextInt();
        } while(input<1 || input>5);
        System.out.println("You entered "+input);
    }
}
```

Simple example

DoWhileCount.java

```
public class DoWhileCount {
   public static void main(String[] args) {
      int count = 0;
      do {
         count++; //short for count=count+1
         System.out.print(count+" ");
      } while(count<20);
   }
}
```

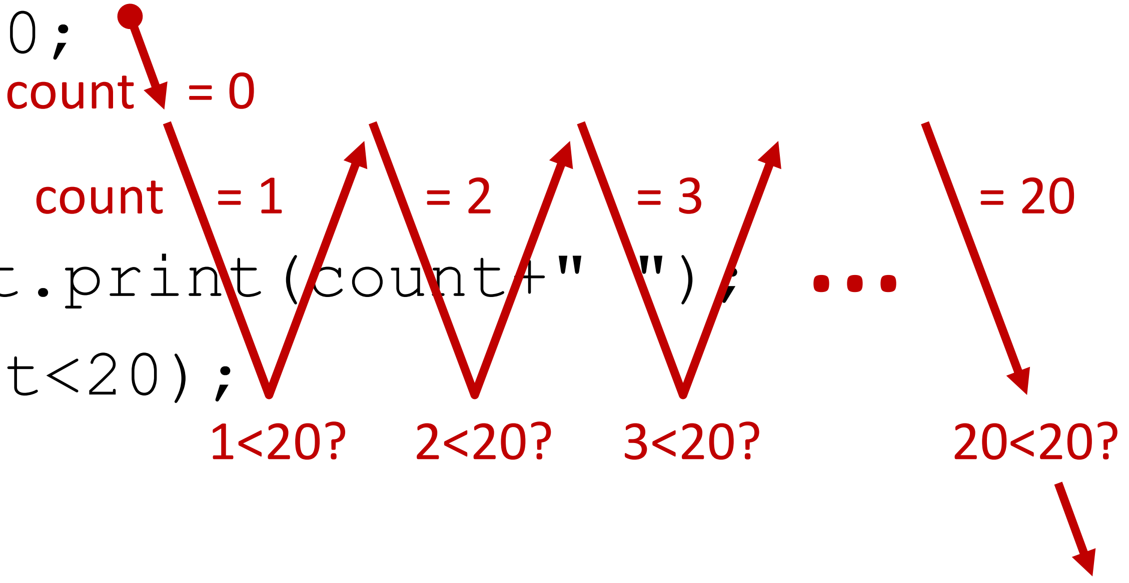
Terminal

```
$ java DoWhileCount
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

Simple example

DoWhileCount.java

```
public class DoWhileCount {
   public static void main(String[] args) {
      int count = 0;
      do {
         count++;
         System.out.print(count+" ");
      } while(count<20);
   }
}
```



Terminal

```
$ java DoWhileCount
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

Simple example

DoWhileCount.java

```
public class DoWhileCount {
    public static void main(String[] args) {
        int count = 0;
        do {
            count++; //short for count=count+1
            System.out.print(count+" ");
        } while(count<20);
    }
}
```

Note the use of `print` (rather than `println`) here. This doesn't print out a new line character at the end.

Terminal

```
$ java DoWhileCount
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

Simple example

```
int count = 0;
do {
   count++;
   System.out.print(count+" ");
} while(count<20);
```

```
int count = 0;
while(count<=20) {
   count++;
   System.out.print(count+" ");
}
```

These two loops do the same thing, but note the small but important difference in the conditions

Simple example

```
int count = 0;
do {
   count++;
   System.out.print(count+" ");
} while(count<20);
```

These two loops do the same thing, but note the small but important difference in the conditions

```
int count = 0;
while(count<=20) {
   count++;
   System.out.print(count+" ");
}
```

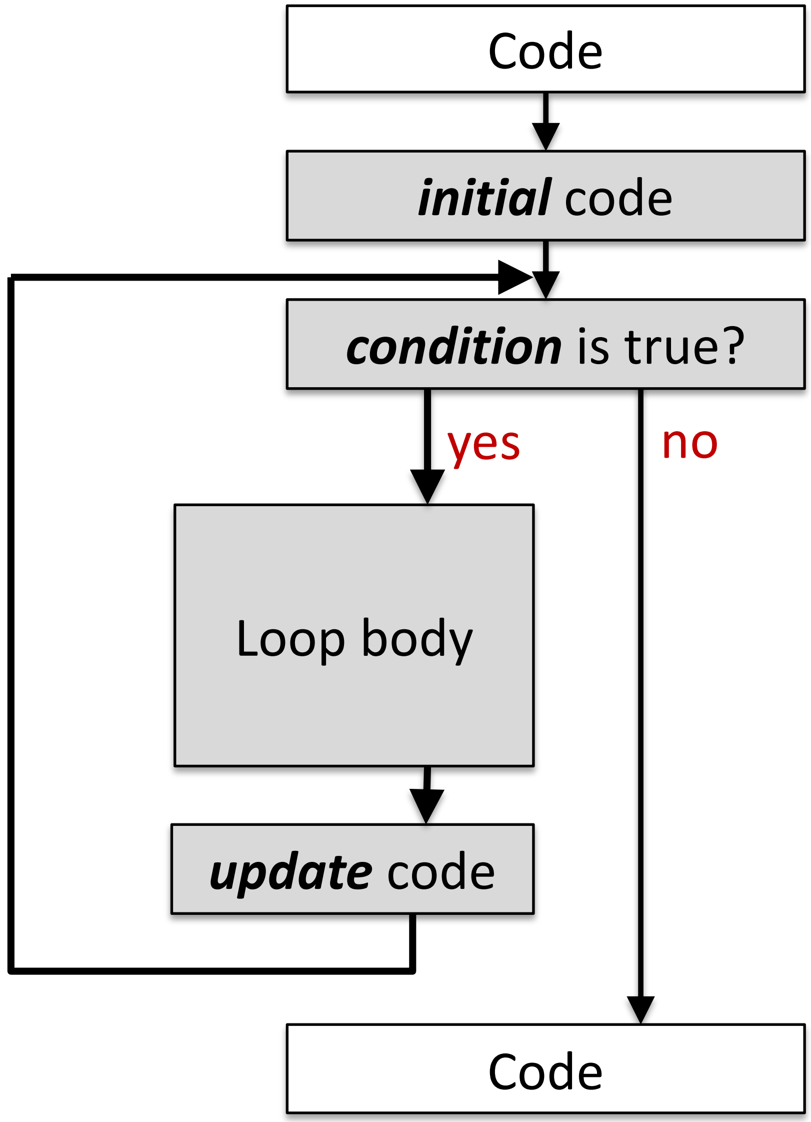
Note also that the behaviour is based on the value of a single variable, i.e. `count`

`for` loops

```
for( initial; condition; update ) {
    // loop body
}
```

- *`initial`* specifies the initial value of one or more loop variables
- *`condition`* specifies the condition where the loop will continue iterating
- *`update`* specifies how the loop variables are to be modified at the end of each iteration

Control Flow



```
// code

for(initial;
      condition;
      update) {
   // loop body
}


// code
```

`for` loops

We can make the previous program shorter, and more readable, by using a `for` loop:

ForCount.java

```
public class ForCount {
   public static void main(String[] args) {
      for(int count=0; count<=20; count++) {
         System.out.print(count+" ");
      }
   }
}
```

These are often used when the condition is based on the value of a numeric variable (**loop variable**) whose value changes each iteration

Any Questions?

Exercise

Using a loop, write a program to print out the children's song "Ten Green Bottles"

10 green bottles standing on the wall
10 green bottles standing on the wall
And if 1 green bottle should accidentally fall
There will be **9** green bottles standing on the wall

9 green bottles standing on the wall
9 green bottles standing on the wall
And if 1 green bottle should accidentally fall
There will be **8** green bottles standing on the wall

… until you reach 0 green bottles

Example: Multiplication Table

We want a program that prints out the multiplication table for a particular value, e.g.

```
Multiplication table for 5
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
```

Example: Multiplication Table

We want a program that prints out the multiplication table for a particular value, e.g.

```
Multiplication table for 5
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
```

The fact that we're repeating the same type of calculation over and over again suggests that we could use a loop for this.

Example: Multiplication Table

We want a program that prints out the multiplication table for a particular value, e.g.

```
Multiplication table for 5
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
```

The fact that we have something that is counting up suggests a `for` loop.

Multiplication Table

MultiplicationTable.java

```
public class MultiplicationTable {
   public static void main(String[] args) {
      // variable declarations
      int table;     // multiplicand
      int maxvalue; // largest multiplier
      int product;  // multiplier x multiplicand

      // get input from user
      Scanner scan = new Scanner(System.in);
      System.out.println("What number would you like
                  to produce a table for?");
      table = scan.nextInt();
      System.out.println("What is the maximum
                  multiplier for the table?");
      maxvalue = scan.nextInt();
```

Multiplication Table

```
        // output table
        System.out.println(
                "Multiplication table for "+table);

        for(int value=1; value<=maxvalue; value++) {
            product = table * value;
            System.out.println(
                    table+" x "+value+" = "+product);
        }
    }
}
```

Terminal

```
What number would you like to produce a table for? 5
What is the maximum multiplier for the table? 3
Multiplication table for 5
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
```

These are the values of `value` on subsequent iterations of the loop

Part II
More Complex Loops

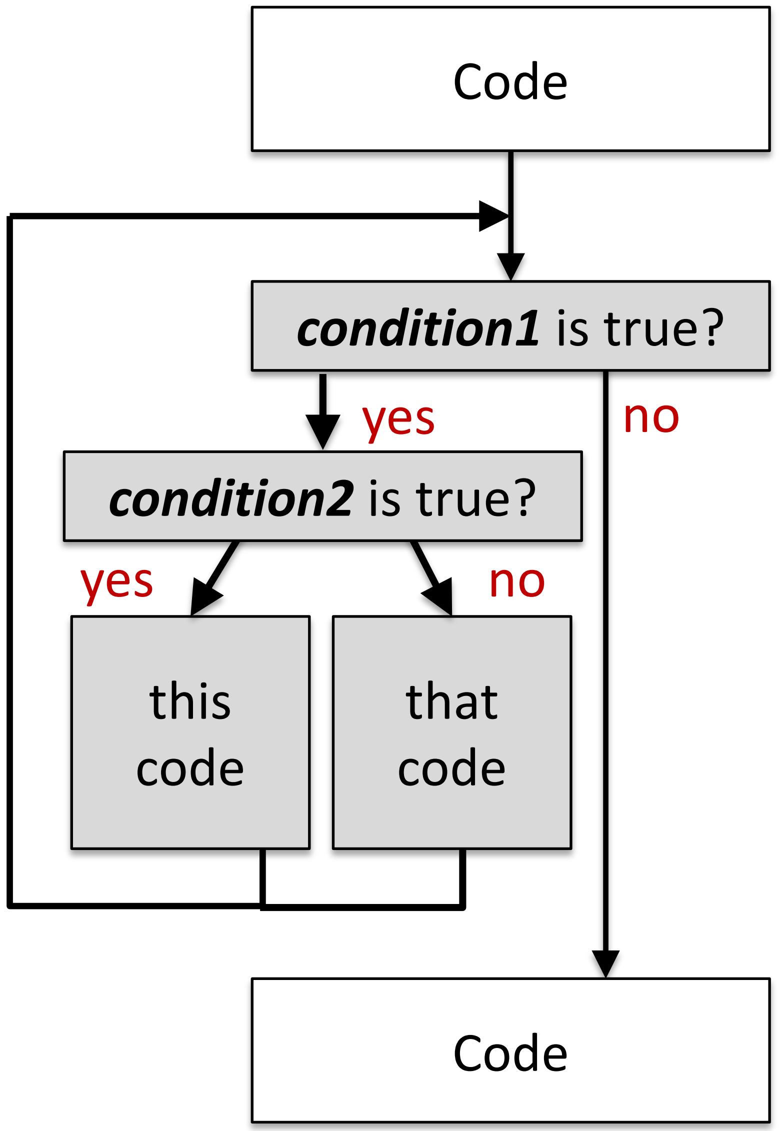
Combining loops with `if`s

Sometimes you want to do different things each time around the loop

- One way of doing this is to embed `if` statements within the loop body, e.g.

```
while(condition1) {
   if(condition2)
      // do something
   else
      // do something else
}
```

Control Flow



```
// code


while(condition1) {
    if(condition2) {
        // this code
    }
    else
        // that code
    }
}

// code
```

Simple Example

EvenOdd.java

```java
public class EvenOdd {
   public static void main(String[] args) {
      for(int num=1; num<=5; num++) {
         if(num % 2 == 0) // % modulus operator
            System.out.println(num+" is even");
         else
            System.out.println(num+" is odd");
      }
   }
}
```

Terminal

```
1 is odd
2 is even
3 is odd
4 is even
5 is odd
```

Nested Loops

To achieve more complex behaviours, it is also possible to nest loops inside other loops, e.g.

NestedForLoop.java

```
public class NestedForLoop {
   public static void main(String[] args) {
      for(int i=1; i<=3; i++)
         for(int j=1; j<=5; j++)
            System.out.print(""+i+","+j+" ");
   }
}
```

Terminal

```
$ java NestedForLoop
1,1 1,2 1,3 1,4 1,5 2,1 2,2 2,3 2,4 2,5 3,1 3,2 3,3 3,4 3,5
```

Nested Loops

To achieve more complex behaviours, it is also possible to nest loops inside other loops, e.g.

NestedForLoop.java

```
public class NestedForLoop {
  public static void main(String[] args) {
    for(int i=1; i<=3; i++) Outer loop
      for(int j=1; j<=5; j++) Inner loop
        System.out.print(""+i+","+j+" ");
  }
}
```

Terminal

```
$ java NestedForLoop
1,1 1,2 1,3 1,4 1,5 2,1 2,2 2,3 2,4 2,5 3,1 3,2 3,3 3,4 3,5
```

$\underbrace{\texttt{1,1 1,2 1,3 1,4 1,5}}_{\text{Outer i=1}}\ \underbrace{\texttt{2,1 2,2 2,3 2,4 2,5}}_{\text{Outer i=2}}\ \underbrace{\texttt{3,1 3,2 3,3 3,4 3,5}}_{\text{Outer i=3}}$

Multiplication Table

```
        // output table
        System.out.println(
              "Multiplication table for "+table);

        for(int value=1; value<=maxvalue; value++) {
           product = table * value;
           System.out.println(
                 table+" x "+value+" = "+product);
        }
    }
}
```

Q: How would we modify this code so that it produces all multiplication tables up to a given number?

Multiplication Table

```
        // output table
        System.out.println(
              "Multiplication table for "+table);

        for(int value=1; value<=maxvalue; value++) {
           product = table * value;
           System.out.println(
                 table+" x "+value+" = "+product);
        }
    }
}
```

A: We need to run this code multiple times for different values of `table`. We can do this with an **outer loop**.

Multiplication Tables

MultiplicationTables.java

```java
public class MultiplicationTables {
   public static void main(String[] args) {
      // variable declarations
      int maxtable; // largest multiplicand
      int maxvalue; // largest multiplier
      int product;  // multiplier x multiplicand

      // get input from user
      Scanner scan = new Scanner(System.in);
      System.out.println("What number would you like
            to produce tables up to?");
      maxtable = scan.nextInt();
      System.out.println("What is the maximum
            multiplier for each table?");
      maxvalue = scan.nextInt();
```

Multiplication Tables

```
        // output tables
        for(int table=1; table<=maxtable; table++) {
            System.out.println(
                  "Multiplication table for "+table);
            for(int value=1; value<=maxvalue; value++) {
                product = table * value;
                System.out.println(
                    table+" x "+value+" = "+product);
            }
        }
    }
}
```

The block highlighted in red is the **outer loop**.
The original for loop is now referred to as an **inner loop**.

Multiplication Tables

Terminal

```
$ java MultiplicationTables
What number would you like to produce tables up to? 3
What is the maximum multiplier for each table? 4
Multiplication table for 1
1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
1 x 4 = 4

Multiplication table for 2
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8

Multiplication table for 3
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
```

These are the values of `table` on subsequent iterations of the outer loop

Any Questions?

Escaping From Loops

Sometimes you want your program to leave the loop before it gets to the end of an iteration

- The **`break`** statement can be used to exit the loop at any point
- The **`continue`** statement can be used to move immediately to the next iteration

! But these should be used carefully, since they sometimes make code harder to read

Escaping From Loops

However, they sometimes increase code clarity

```
public class SuperstitiousCount {
  public static void main(String[] args) {
    for(i=1; i<=20; i++) {
      if(i!=12+1) {
        System.out.print(i+" ");
      }
    }
  }
}
```

Escaping From Loops

However, they sometimes increase code clarity

```
public class SuperstitiousCount {
    public static void main(String[] args) {
        for(i=1; i<=20; i++) {
            if(i==12+1) continue;
            System.out.print(i+" ");
        }
    }
}
```

In this example, we've removed the need for extra indentation. Too much indentation can be confusing.

Escaping From Loops

`break` is useful when a loop has two conditions. Here, it exits either after 10 repeats or when the user enters 0:

SumNumbers.java

```
public class SumNumbers {
    public static void main(String[] args) {
        int input, sum = 0;
        System.out.println("Enter 10 numbers
                        (enter 0 to finish early)");
        Scanner scan = new Scanner(System.in);
        for(int i=0; i<10; i++) {
            input = scan.nextInt();
            if(input==0) break;
            sum += input; // add input to sum
        }
        System.out.println("Total: "+sum);
    }
}
```

Java `for` loops

```
public class ForCount {
   public static void main(String[] args) {
      for(int count=0; count<=20; count++) {
         System.out.println(count+" ");
      }
   }
}
```

There's a lot of flexibility in the syntax of Java loops.

```
public class ForCount {
   public static void main(String[] args) {
      int count;
      for(count=0; count<=20; count++) {
         System.out.println(count+" ");
      }
   }
}
```

You don't have to declare the loop variable in the for statement – you can use an **existing variable**.

Java `for` loops

```
public class ForCount {
   public static void main(String[] args) {
      for(int count=0; count<=20; count++) {
         System.out.println(count+" ");
      }
   }
}
```

There's a lot of flexibility in the syntax of Java loops.

```
public class ForCount {
   public static void main(String[] args) {
      int count = 0;
      for(; count<=20; count++) {
         System.out.println(count+" ");
      }
   }
}
```

You can also miss out the initialisation part, and use the **existing value** of the existing variable.

Java `for` loops

In fact, all the parts of the `for` statement are optional in Java. So, this is valid:

```
public class ForCount {
   public static void main(String[] args) {
      for(;;) {
         System.out.println("Loop forever!");
      }
   }
}
```

Though `while(true)` would be clearer in this case. Most of the time when you use for, each part of the for loop will be used.

Java `for` loops

`for` loops can count in more complicated ways:

- They can count down as well as up, e.g.

```
for(int i=10; i>0; i--) //i=i-1
```

- They don't have to go up or down by 1, e.g.

```
for(double i=0; i<100; i=i+0.5)
for(int i=0; i<=100; i+=10) //i=i+10
for(int i=2; i<=256; i*=2)  //i=i*2
```

Java `for` loops

`for` loops can count in more complicated ways:

- You can use multiple loop variables and conditions involving multiple variables, e.g.

```
for(i=0,j=2; i<100&&j<256; i+=10,j*=2)

for(i=100,j=1; i>0; i=i-j,j++)
```

Complicated Counting

ComplicatedCount.java

```java
public class ComplicatedCount {
   public static void main(String[] args) {
      int i, j;
      for(i=100, j=1; i>0; i=i-j, j++) {
         System.out.print(i+" ");
      }
   }
}
```

Terminal

```
$ java ComplicatedCount
100 99 97 94 90 85 79 72 64 55 45 34 22 9
```

Any Questions?

Loop Exercises

Write a program to print a square pattern of a given size, e.g. 5:

```
*  *  *  *  *
*  *  *  *  *
*  *  *  *  *
*  *  *  *  *
*  *  *  *  *
```

Loop Exercises

Write a program to print a square pattern of a given size, e.g. 5:

```
*  *  *  *  *
*  *  *  *  *
*  *  *  *  *
*  *  *  *  *
*  *  *  *  *
```

```
public class CheatyAnswer {
    public static void main(String[] args) {
        System.out.println("* * * * *");
        System.out.println("* * * * *");
        System.out.println("* * * * *");
        System.out.println("* * * * *");
        System.out.println("* * * * *");
    }
}
```

Loop Exercises

Write a program to print a square pattern of a given size, e.g. 5:

```
*  *  *  *  *
*  *  *  *  *
*  *  *  *  *
*  *  *  *  *
*  *  *  *  *
```

```java
public class CheatyAnswer {
    public static void main(String[] args) {
        System.out.println("* * * * *");
        System.out.println("* * * * *");
        System.out.println("* * * * *");
        System.out.println("* * * * *");
        System.out.println("* * * * *");
    }
}
```

Hint: you need to use a loop within a loop

Loop Exercises

Write a program to print the following pattern:

```
*
*  *
*  *  *
*  *  *  *
*  *  *  *  *
*  *  *  *  *  *
*  *  *  *  *  *  *
*  *  *  *  *  *  *  *
```

Loop Exercises

Now modify it to print the following pattern:

```
*
+ +
* * *
+ + + +
* * * * *
+ + + + + +
* * * * * * *
+ + + + + + + +
```

Loop Exercises

How about?

```
*
*  *
*     *
*        *
*           *
*              *
*                 *
*  *  *  *  *  *  *  *
```

Loop Exercises

And finally:

```
       *
      * *
     *   *
    *     *
   *       *
  *         *
 *           *
* * * * * * * *
```

Next Week's Lab

You'll be doing some exercises involving iteration

- Please use Eclipse for these

Next Lecture

- Arrays
  - These are a loop's best friend, or vice versa
  - So, make sure you're up to speed with loops

Summary

- Java provides three forms of iteration statement
- `while`, `do…while` and `for`
- `for` is used when the loop's termination condition is based on the value of a loop variable
- Loops and conditional execution statements can be nested for more complex behaviour
- `continue` and `break` can be used to finish an iteration or loop early