Data Structures & Algorithms 1

Topic 2 – Programming Revision

Programming language

- We will need to use some programming language to represent data structures and algorithms

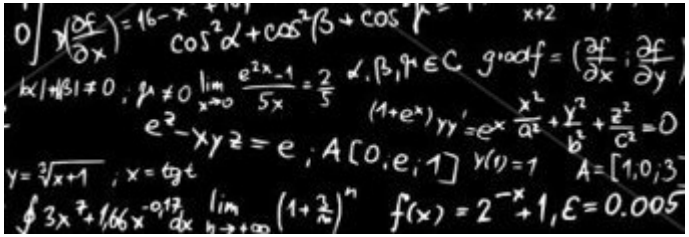
- We will use the Java language

- However, you could use any other programming language to encode the same ideas - another popular language is C++

Programming Languages

- Languages are on a continuum from low-level electronics to high-level
- At the lowest level the programming language provides no abstraction from the physical device
- At the highest level the language is so abstract it is purely mathematical
- Java is in the middle

Haskell
Lisp



Python
Ruby
Perl

Java
C#

C++
C

Assembly
language

Electronic circuits

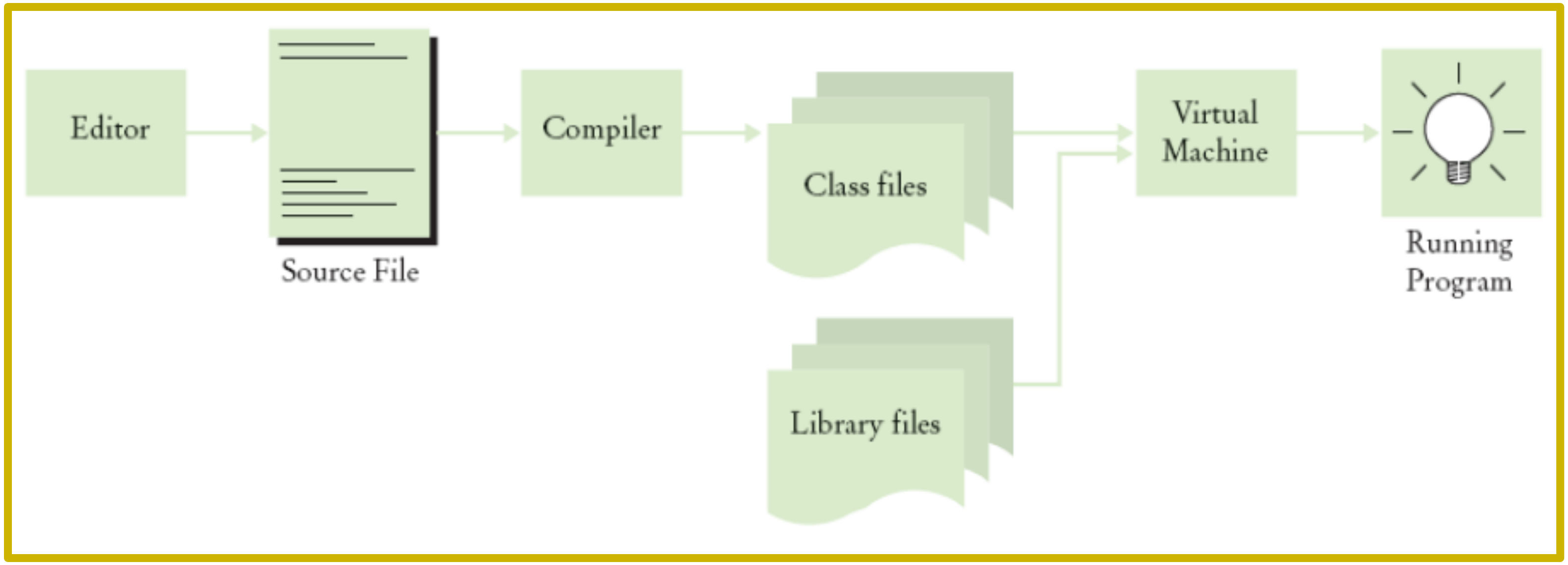
Java programming

- Java is a programming language first released in 1995 originally developed by James Gosling at Sun Microsystems
- One reason Java is popular is because it is platform independent
- Programs written in Java can run on any hardware or operating-system
- Compiled code is run on a Java Virtual Machine (JVM) which converts it to the native language

Platform independence

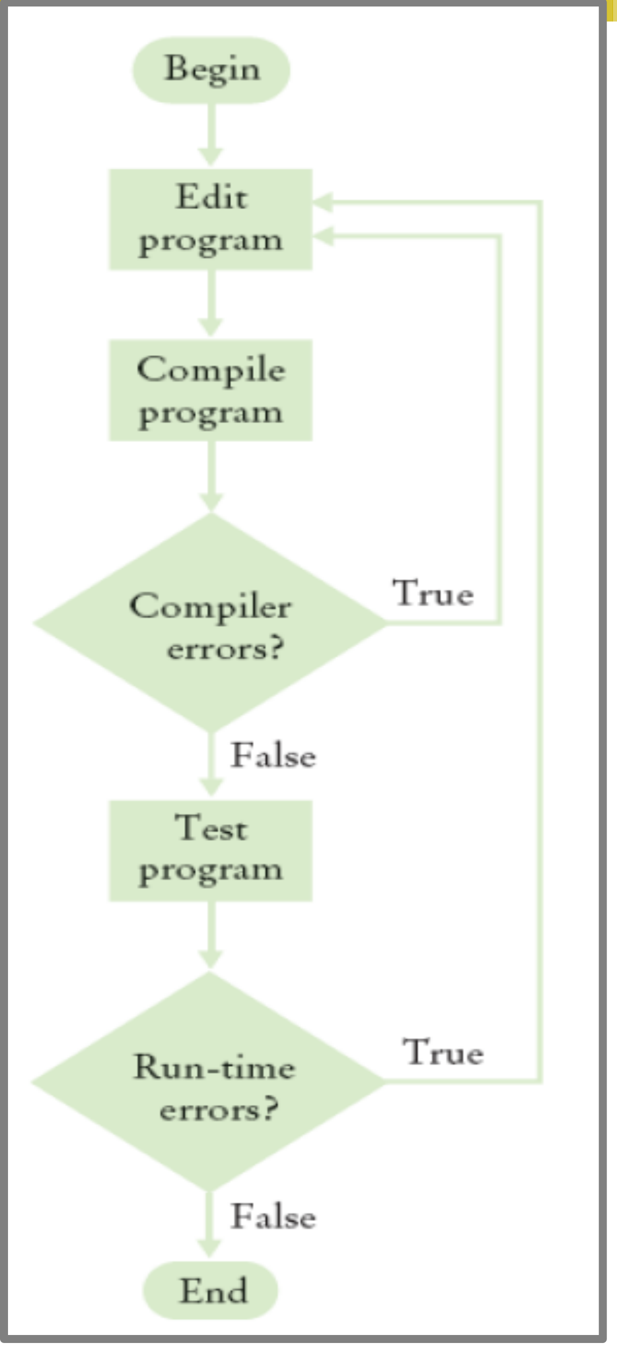
- Turing showed that machine, software and input can all be represented in terms of patterns of information
- The compiler translates the Java code into machine code that the JVM can run
- The JVM is a machine simulated by the actual physical machine it is running on

The compilation process



Edit, compile, run

- Compiling turns the code you wrote in Java (.java file) into a format that the computer can run on the JVM (.class file)
- You can't run your code without compiling it
- Every time you change your code you need to recompile



Revision

- We will now revise the following:

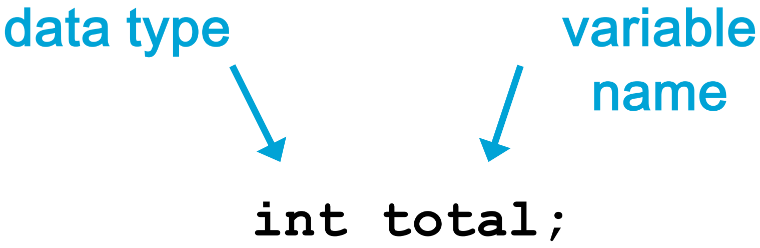
  - Variables & Data Types: (ints, doubles)
  - Variable Operators: (addition, subtraction)
  - Selection: (if, else)
  - Iteration: (for, while, do)

Variables

- Variable is a name for a location in memory

- 2 types of variables
  - Primitive (e.g. int and double – usually smaller case letters)
  - Reference (e.g. objects – usually starts with capital letter)

- Must have a type and a name
  - Cannot be a reserved word (public, void, static, int, …)



data type

variable name

```
int total;
```

Variables

- A variable can be given an initial value in the declaration

```
int sum = 0;
int base = 32, max = 149;
```

- When a variable is not initialized, the value of that variable is undefined

Scope & garbage collection

- Variables defined within a member function are local to that function (this is referred to as the scope of a variable)

```
for (int i = 0; i < 50; i++){…}
```

- Local variables are destroyed (garbage collected) when function exits (or goes out of scope.)
- Programmer need not worry about de-allocating memory for out of scope objects/variables.
  - Unlike in C or C++

Assignment

- An *assignment statement* changes the value of a variable
- The assignment operator is the = sign

```
total = 55;
```

- The expression on the right is evaluated and the result is stored in the variable on the left
- The value that was in `total` is overwritten
- You can assign only a value to a variable that is consistent with the variable's declared type

Primitive types

- There are exactly eight primitive data types in Java
- Four of them represent integers:
  - `byte, short, int, long`
- Two of them represent floating point numbers:
  - `float, double`
- One of them represents characters:
  - `char`
- And one of them represents true/false boolean values:
  - `boolean`

Bits and bytes

- A single bit is a one or a zero, a true or a false, a "flag" which is on or off
- A byte is made up of 8 bits like this : 10110001
- 1 Kilobyte = about 1,000 bytes (1,024 to be precise)
- 1 Megabyte = about 1,000,000 bytes (1,024 * 1,024)
- 1 Gigabyte = about 1,000,000,000 bytes

Primitive types

Type	Description	Size
`int`	The integer type, with range –2,147,483,648 . . . 2,147,483,647	4 bytes
`byte`	The type describing a single byte, with range –128 . . . 127	1 byte
`short`	The short integer type, with range –32768 . . . 32767	2 bytes
`long`	The long integer type, with range – 9,223,372,036,854,775,808 . . . –9,223,372,036,854,775,807	8 bytes

Primitive types

Type	Description	Size
`double`	The double-precision floating-point type, with a range of about $\pm 10^{308}$ and about 15 significant decimal digits	8 bytes
`float`	The single-precision floating-point type, with a range of about $\pm 10^{38}$ and about 7 significant decimal digits	4 bytes
`char`	The character type, representing code units in the Unicode encoding scheme	2 bytes
`boolean`	The type with the two truth values `false` and `true`	1 bit

Number types

- Illegal to assign a floating-point expression to an integer variable

```
double balance = 13.75;
int dollars = balance; // Error
```

- `Casts`: used to convert a value to a different type

```
int dollars = (int) balance; // OK
```

- `Math.round` converts a floating-point number to nearest integer

```
long rounded = Math.round(balance);
// if balance is 13.75, then
// rounded is set to 14
```

Arithmetic expressions

- *Arithmetic expressions* compute numeric results and make use of the arithmetic operators:

Addition	+
Subtraction	-
Multiplication	*
Division	/
Remainder	%

- If either or both operands associated with an arithmetic operator are floating point, the result is a floating point

Modulus operator %

- The % symbol is the modulus operator
- This divides the first number by the second number and gives you the remainder
  - 55 % 10 = 5
  - 42 % 4 = 2

Answer

- Both of these work

- How can we figure out how many times 7 divides into a variable called ***number***?

  - (number - (number % 7))/ 7
  - number / 7 – ((number / 7) % 1)

Operator precedence

- Operators can be combined into complex expressions

```
result  =  total + count / max - offset;
```

- Multiplication, division, and remainder are evaluated prior to addition, subtraction, and string concatenation (BOMDAS rule)
- Arithmetic operators with the same precedence are evaluated from left to right
- Parentheses can be used to force the evaluation order

Increment and decrement

- The increment and decrement operators are arithmetic and operate on one operand
- The *increment operator* (`++`) adds one to its operand
- The *decrement operator* (`--`) subtracts one from its operand
- The statement `count++;`

  is functionally equivalent to `count = count + 1;`

Assignment operators

- Often we perform an operation on a variable, and then store the result back into that variable
- Java provides *assignment operators* to simplify that process
- For example, the statement

```
num += count;
```

is equivalent to

```
num = num + count;
```

Relational operators

>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
==	equal to
!=	not equal to

Frequent mistake!!

- If we want to put the variable “number” equal to ten we use one equals sign
  - number = 10;
- However, if we want to check *if* number is equal to ten then we use a double equals
  - if (number == 10)

The `Math` class

- Math class: contains methods like `sqrt` and `pow`
- To compute x^n, you write `Math.pow(x, n)`
- However, to compute x^2 it is significantly more efficient simply to compute `x * x`
- To take the square root of a number, use the `Math.sqrt`; for example, `Math.sqrt(x)`

The Math class

- In Java,

$$\boxed{\frac{-b+\sqrt{b^2-4ac}}{2a}}$$

can be represented as

```
(-b + Math.sqrt(b * b - 4 * a * c)) / (2 * a)
```

Mathematical methods in Java

`Math.sqrt(x)`	square root
`Math.pow(x, y)`	power x^y
`Math.exp(x)`	e^x
`Math.log(x)`	natural log

`Math.sin(x), Math.cos(x),`
`Math.tan(x)`
	sine, cosine, tangent (x in radian)

`Math.round(x)`	closest integer to x

`Math.min(x, y)`
`Math.max(x, y)`
	minimum, maximum

Questions

- What is the value of `643 / 100`?
  - Depends on whether *double* or *int*
- What is the value of `643 % 100`?
  - 43
- Why doesn't the following statement compute the average of `s1`, `s2`, and `s3`?
  - Missing brackets

```
double average = s1 + s2 + s3 / 3; // Error
```

Strings

- A string is a sequence of characters
- Strings are objects of the `String` class
- String variables:

```
String message = "Hello, World!";
```

- String length:

```
int n = message.length();
```

- Empty string:

```
""
```

Concatenation

- Use the + operator:

```
String name = "Dave";
String message = "Hello, " + name;
   // message is "Hello, Dave"
```

- If one of the arguments of the + operator is a string, the other is converted to a string

```
String a = "Agent";
int n = 7;
String bond = a + n; // bond is Agent7
```

Concatenation when printing

- Useful to reduce the number of `System.out.print` instructions

```
System.out.print("The total is ");
System.out.println(total);
```

versus

```
System.out.println("The total is " + total);
```

Converting between Strings and numbers

- Convert to number:

```
int n = Integer.parseInt(str);
double x = Double.parseDouble(str);
```

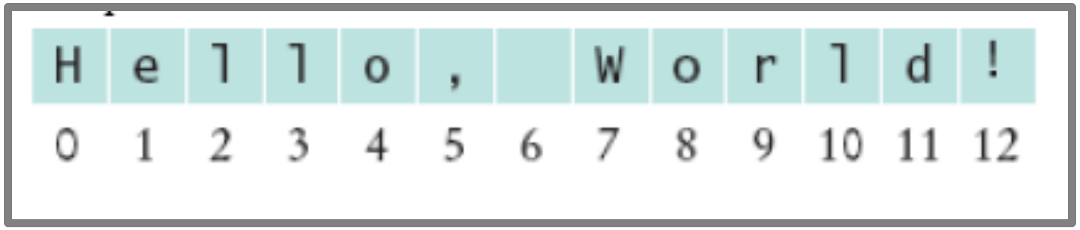
- Convert to string:

```
String str = "" + n;
str = Integer.toString(n);
```

Substrings

```
String greeting = "Hello, World!";
String sub = greeting.substring(0, 5); // sub is "Hello"
```

- Supply start and stopping index
- First position is at 0



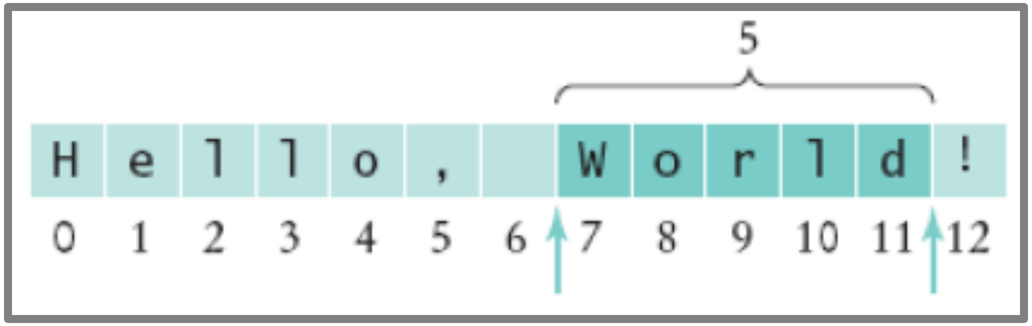
H	e	l	l	o	,		W	o	r	l	d	!
0	1	2	3	4	5	6	7	8	9	10	11	12

String Positions

Substrings

- Syntax is (start index, stopping index)
- Stops before it gets to the stopping index
- Substring length is ‘ending index – stopping index’

greeting.substring(7, 12) :



H	e	l	l	o	,		W	o	r	l	d	!
0	1	2	3	4	5	6	7	8	9	10	11	12

Extracting a Substring

Questions

1. Assuming the `String` variable `s` holds the value `“Hello"`, what is the effect of the assignment `s = s + s.length()`?

2. Assuming the `String` variable `college` holds the value `"Maynooth"`, what is the value of `college.substring(1, 2)`?

3. How about `college.substring(2, college.length() - 3)`?

Answers

1. s is set to the string Hello5

2. The string “a”

3. The string “yno”

charAt( )

- Another handy method that comes with Strings is charAt( )
- This allows us to pick out characters at particular locations in the string
- The first character has position 0

```
String s = “hello”;
System.out.println(s.charAt(0));

h
```

Comparing Strings

- ***Strings are not numbers!!!***
- To test whether two strings are equal you must use a method called equals:

```
    if (string1.equals(string2)) …
```

- Do not use the == operator to compare strings.

```
    if (string1==string2)
```

- The above tests to see if two string variables refer to the same string object – not the same as comparing values

More String comparisons

- The `compareTo` Method compares strings in dictionary order:

- If `s1.compareTo(s2) < 0` then the string s1 comes before the string s2 in the dictionary

- What do the following tell us?
    - `s1.compareTo(s2) == 0`
    - `s1.compareTo(s2) > 0`

Reading input

- `System.in` has minimal set of features–it can only read one byte at a time – not much use

- In Java 5.0, `Scanner` class was added to read keyboard input in a convenient manner

```
Scanner in = new Scanner(System.in);
System.out.print("Enter quantity: ");
int quantity = in.nextInt();
```

Reading input

- `nextDouble` reads a double
- `nextLine` reads a line (until user hits Enter)
- `nextWord` reads a word (until any white space)
- You will need to include this line at the top:

```
import java.util.Scanner;
```

Sequence, selection, iteration

- Almost all programming languages (e.g. Java, C, Pascal, C++, Cobol...) are based on 3 simple structures:
  - Sequence: lines separated by semicolon
  - Selection: if / else
  - Iteration: for/ while/ do

Selection statements

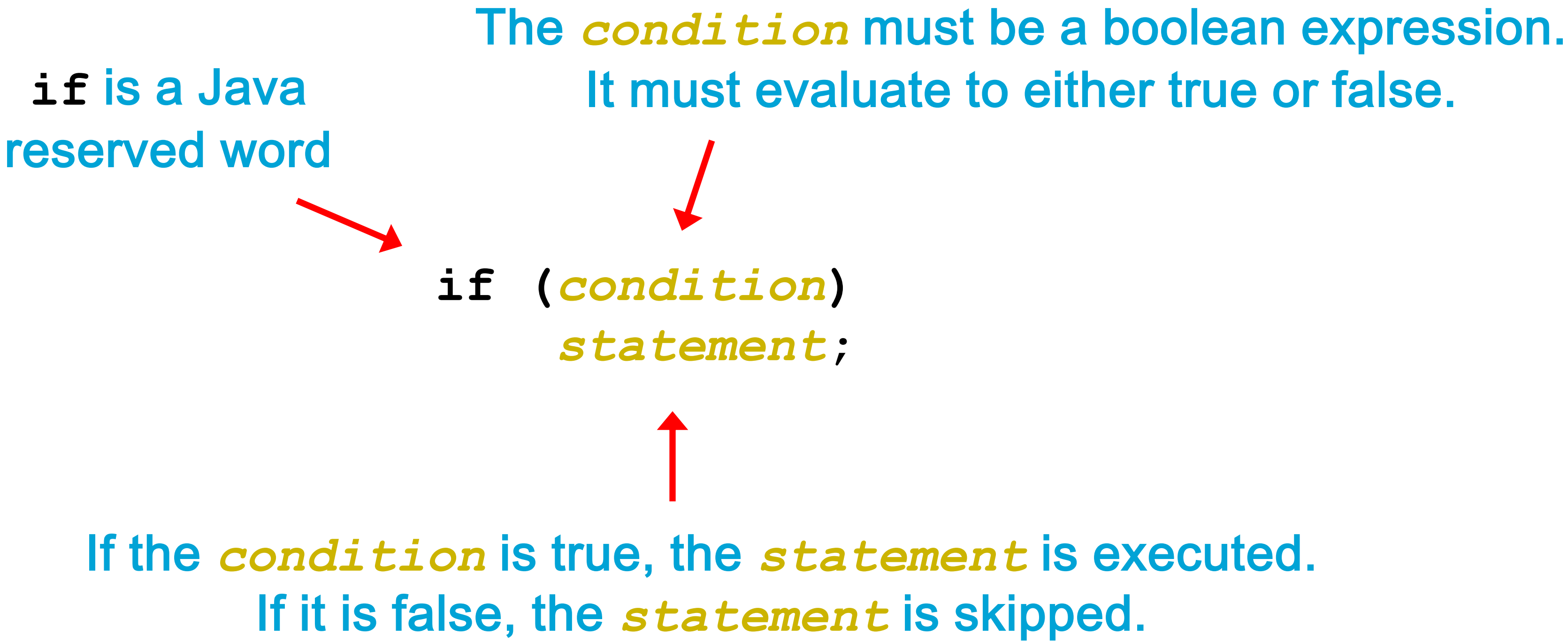
- A *conditional statement* lets us choose which statement will be executed next by using a conditional test
  - the *if statement*
  - the *if-else statement*

- Conditional test is an expression that results in a boolean value using relational operators

- If we have the statement `int x = 3;` the conditional test `(x >= 2)` evaluates to `true`

The if Statement

- The *if statement* has the following syntax:



`if` is a Java reserved word

The *`condition`* must be a boolean expression. It must evaluate to either true or false.

```
if (condition)
    statement;
```

If the *`condition`* is true, the *`statement`* is executed. If it is false, the *`statement`* is skipped.

The if-else Statement

- An *else clause* can be added to an if statement to make an *if-else statement*

```
if ( condition )
   statement1;
else
   statement2;
```

- If the *condition* is true, *statement1* is executed
- If the condition is false, *statement2* is executed
- One or the other will be executed, but not both

Block statements

- Several statements can be grouped together into a *block statement*
- A block is delimited by braces : { … }
- You can wrap as many statements as you like into a block statement

Block statement example

```
if (guess == answer) {

 System.out.println(“You guessed right!”);
 correct++;

} else {

 System.out.println(“You guessed wrong.”);
 wrong++;

}
```

Nested if statements

- The statement executed as a result of an if statement or else clause could be another if statement

- These are called *nested if statements*
- You need to use good indentation to keep track of them

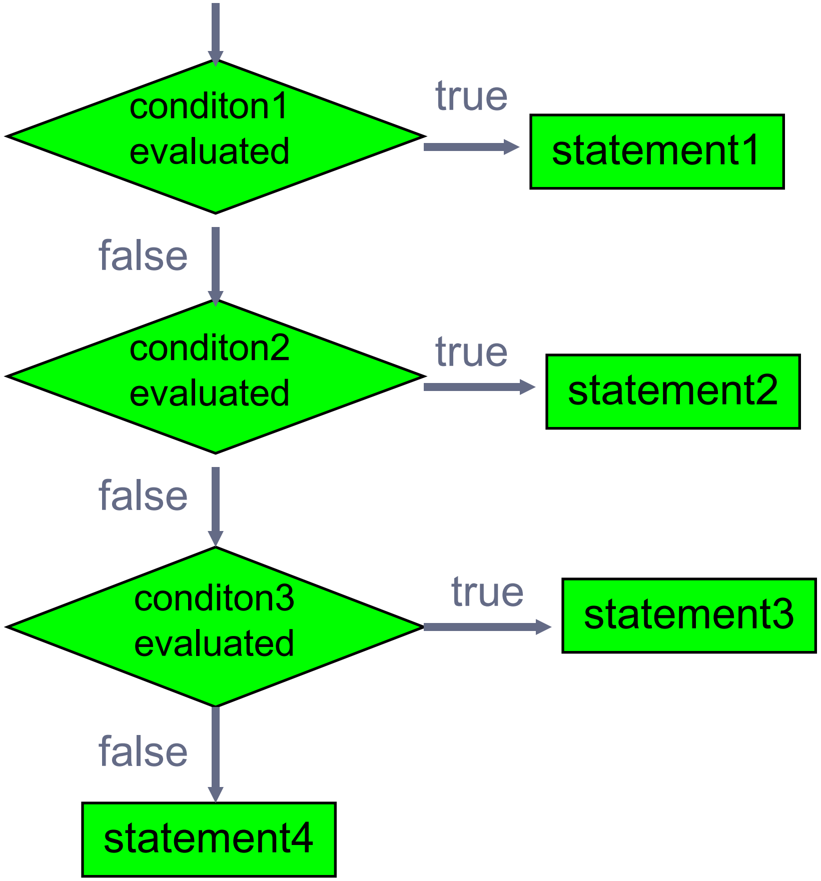
Nested if example

```
if (guess.equals(answer)) {
  if (answer.equals(“yes”)){
     System.out.println(“Yes is correct!”);
  } else {
     System.out.println(“No is correct!”);
  }
} else {
  System.out.println(“You guessed wrong.”);
}
```

Multiway Selection: Else if

- Sometime you want to select one option from several alternatives

```
if (conditon1)
    statement1;
else if (condition2)
    statement2;
else if (condition3)
    statement3;
else
    statement4;
```



Else if example

```
double numberGrade = 83.6;
char letterGrade;

if (numberGrade >= 89.5) {
    letterGrade = ‘A’;
} else if (numberGrade >= 79.5) {
    letterGrade = ‘B’;
} else if (numberGrade >= 69.5) {
    letterGrade = ‘C’;
} else if (numberGrade >= 59.5) {
    letterGrade = ‘D’;
} else {
    letterGrade = ‘F’;
}
System.out.println(“My grade is ” +
                     numberGrade + “, ” + letterGrade);
```

Output:

```
My grade is 83.6, B
```

Logical operators

- Boolean expressions can use the following *logical operators*:

`!`	Logical NOT
`&&`	Logical AND
`||`	Logical OR

- They all take boolean operands and produce boolean results
- Logical NOT is a unary operator
- Logical AND and logical OR are binary operators

Logical NOT

- If some boolean condition `a` is true, then `!a` is false; if `a` is false, then `!a` is true
- Logical expressions can be shown using *truth tables*

`a`	`!a`
true	false
false	true

Logical AND and logical OR

- The *logical AND* expression

  ```
  a && b
  ```

  is true if both `a` and `b` are true, and false otherwise

- The *logical OR* expression

  ```
  a || b
  ```

  is true if `a` or `b` or both are true, and false otherwise

Truth tables

- A truth table shows the possible true/false combinations of the terms
- Since `&&` and `||` each have two operands, there are four possible combinations of conditions `a` and `b`

`a`	`b`	`a && b`	`a || b`
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

Logical operators

- Conditions can use logical operators to form complex expressions

```
if ((total < MAX+5) && !found)
     System.out.println ("Processing…");
```

- Logical operators have precedence relationships among themselves and with other operators
  - relational and arithmetic operators are evaluated first
  - logical NOT is evaluated before AND & OR

Iteration

- *Repetition statements* (a.k.a. *loops*) allow a statement to be executed multiple times
- Like conditional statements, they are controlled by boolean expressions
- Java has three kinds of repetition statements:
  - the *while loop*
  - the *do loop*
  - the *for loop*

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

The while statement

- The *while statement* has the following syntax:

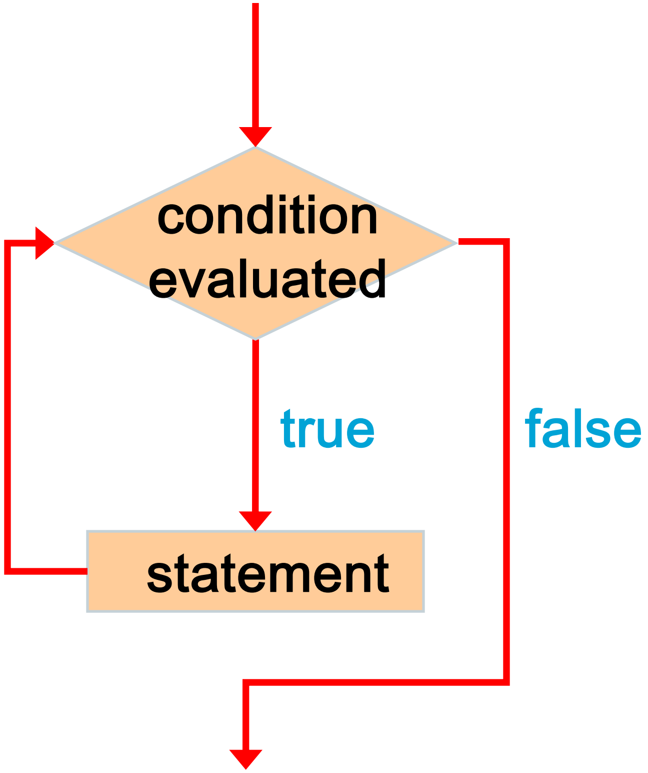
```
while (condition)
    statement;
```

`while` is a reserved word

If the *`condition`* is true, the *`statement`* is executed. Then the *`condition`* is evaluated again.

The *`statement`* is executed repeatedly until the *`condition`* becomes false.

Logic of a while loop



- Note that if the condition of a while statement is false initially, the statement is never executed
- Therefore, the body of a while loop will execute zero or more times

while loop example

```
final int LIMIT = 5;
int count = 1;

while (count <= LIMIT) {

    System.out.println(count);
    count += 1;
}
```

Output:

```
1
2
3
4
5
```

Infinite loops

- The body of a `while` loop eventually must make the condition false
- If not, it is an *infinite loop*, which will execute until the user interrupts the program
- This is a common logical error
- You should always double check to ensure that your loops will terminate normally

The do Statement

- The *do statement* has the following syntax:

`do` and `while` are reserved words

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

The `statement` is executed once initially, and then the `condition` is evaluated

The `statement` is executed repeatedly until the `condition` becomes false

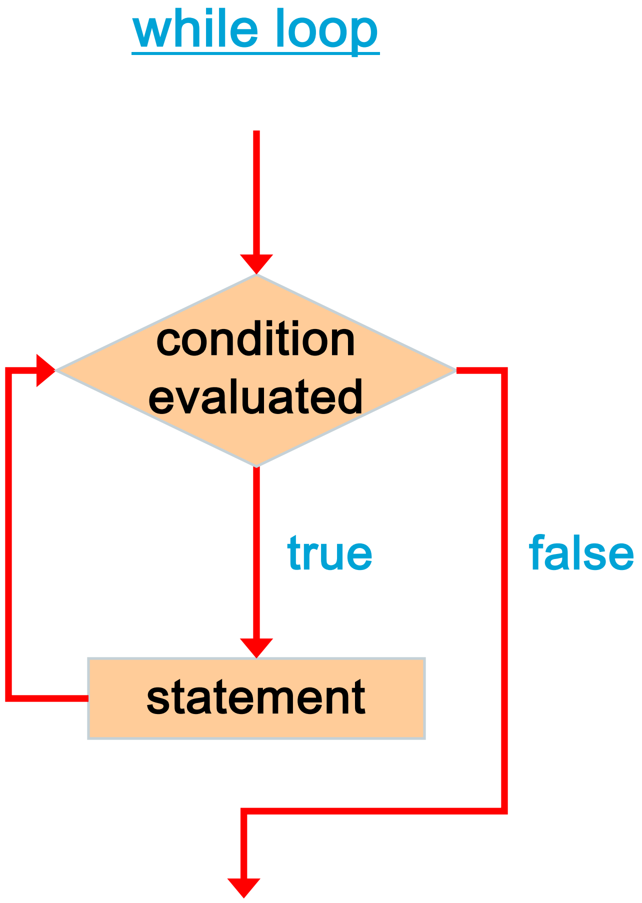
do-while example

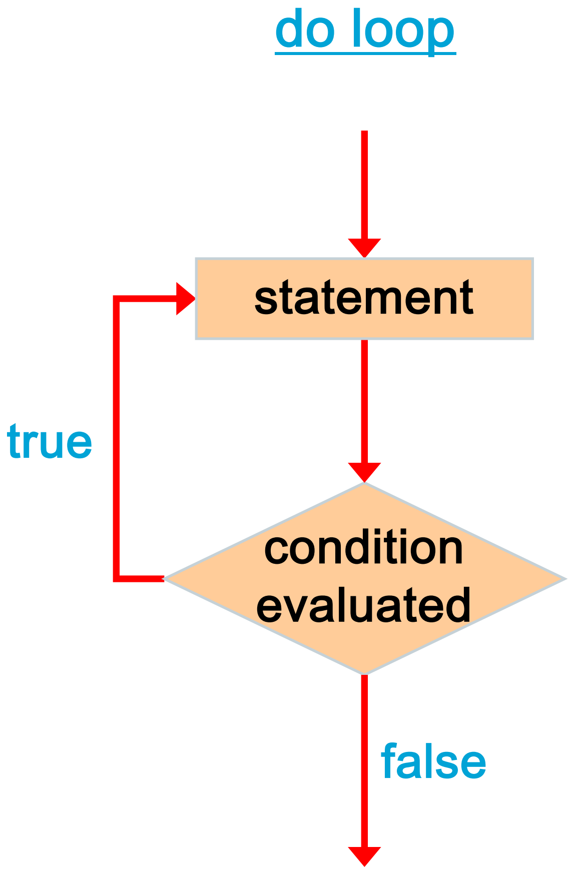
```
final int LIMIT = 5;
int count = 1;

do {
    System.out.println(count);
    count += 1;
} while (count <= LIMIT);
```

Output:

```
1
2
3
4
5
```

Comparing while and do





Nested loops

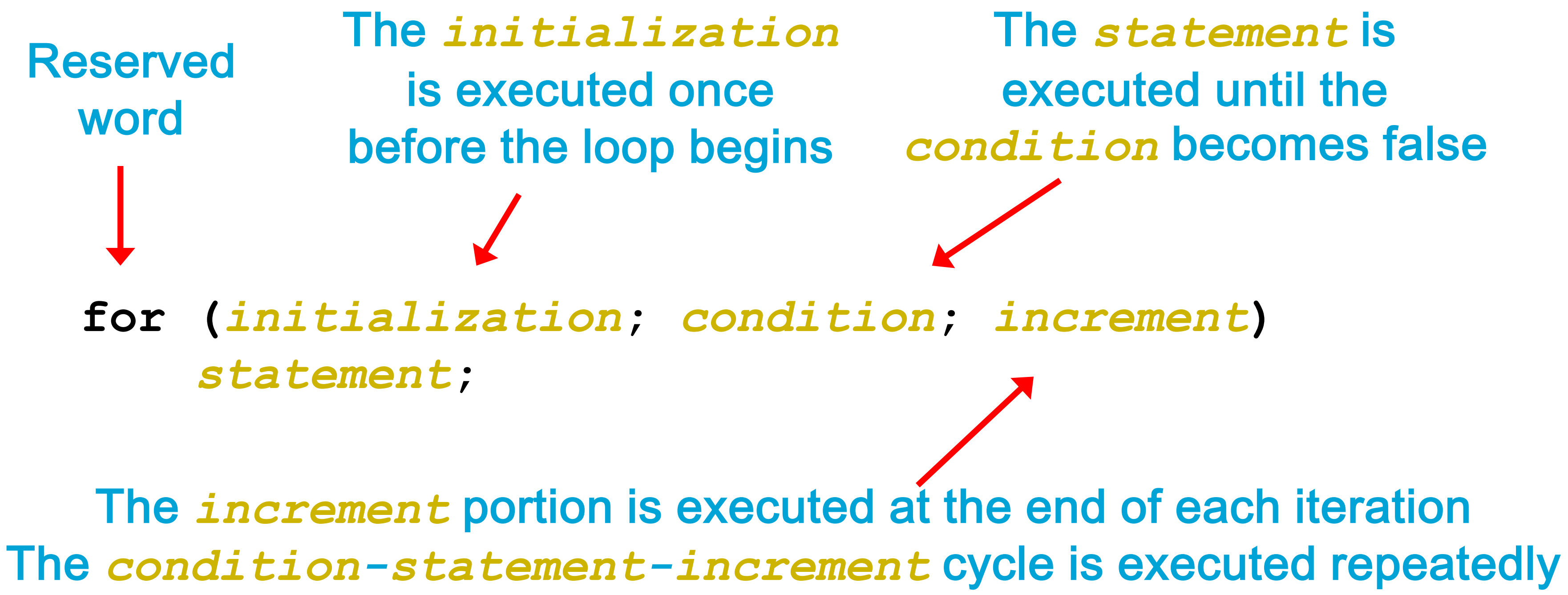
- Similar to nested `if` statements, loops can be nested as well
- For each step of the outer loop, the inner loop goes through its full set of iterations

```
do {
    do {
    } while (…);
while (…);
```

- Don't forget the semicolon after the while!!!

The for Statement

- The *for statement* has the following syntax:



Reserved word

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 *condition-statement-increment* cycle is executed repeatedly

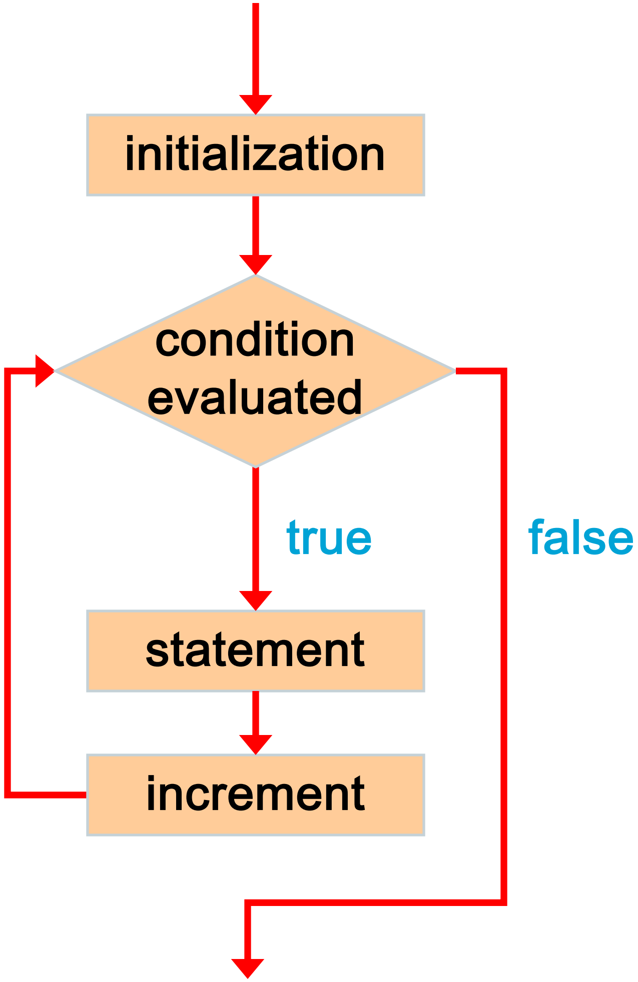
Example

```
for (int i = 0; i < 5; i++) {
    System.out.println(“hello”);
}
```

Output:

```
hello
hello
hello
hello
hello
```

Logic of a for loop



The for statement

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

- Therefore, the body of a `for` loop will execute zero or more times

- It is well suited for executing a loop a specific number of times that can be determined in advance

Example

```
final int LIMIT = 5;
for (int count = 1; count <= LIMIT; count++) {
    System.out.println(count);
}
```

Output:

```
1
2
3
4
5
```

Choosing a loop structure

- When you can’t determine how many times you want to execute the loop body, use a `while` statement or a `do` statement
  - If it might be zero or more times, use a `while` statement
  - If it will be at least once, use a `do` statement
- If you can determine how many times you want to execute the loop body, use a `for` statement