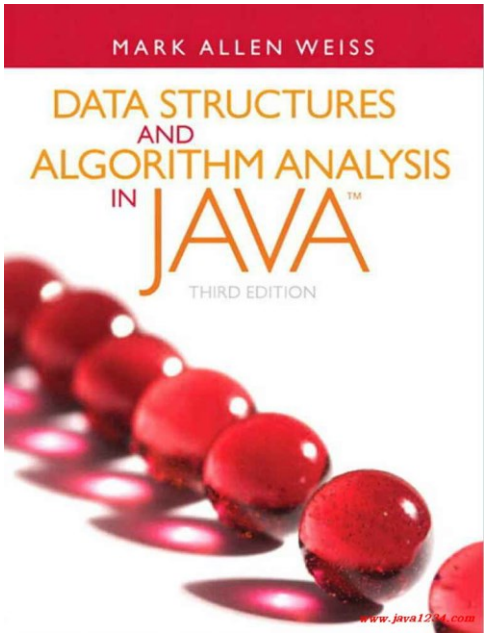
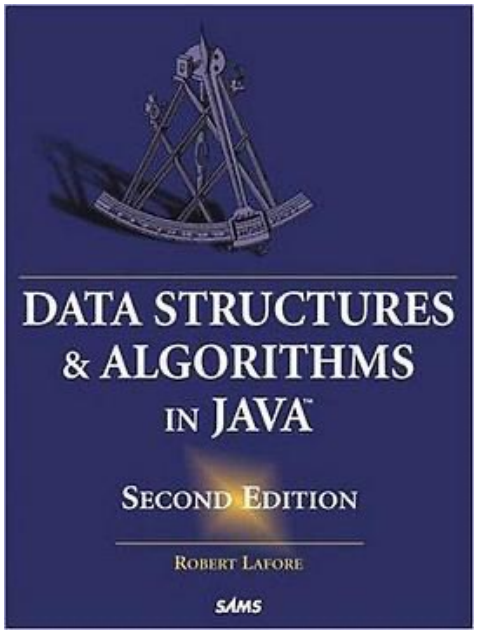
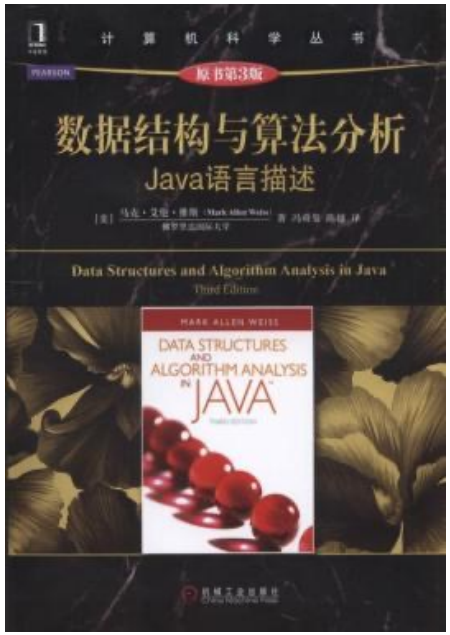
Topic 2 – Programming Revision







Topics

- Introduction
- **Programming Revision**
- Methods and Objects
- Arrays and Array Algorithms
- Big O Notation
- Sorting Algorithms
- Stacks and Queues
- Linked Lists
- Recursion
- Bit Manipulation

Outline

- Programming Language & Java
- Variables & Data Types:
  - int, double
- Variable Operators:
  - addition, subtraction
- Selection:
  - if, else
- Iteration:
  - for, while, do

Outline

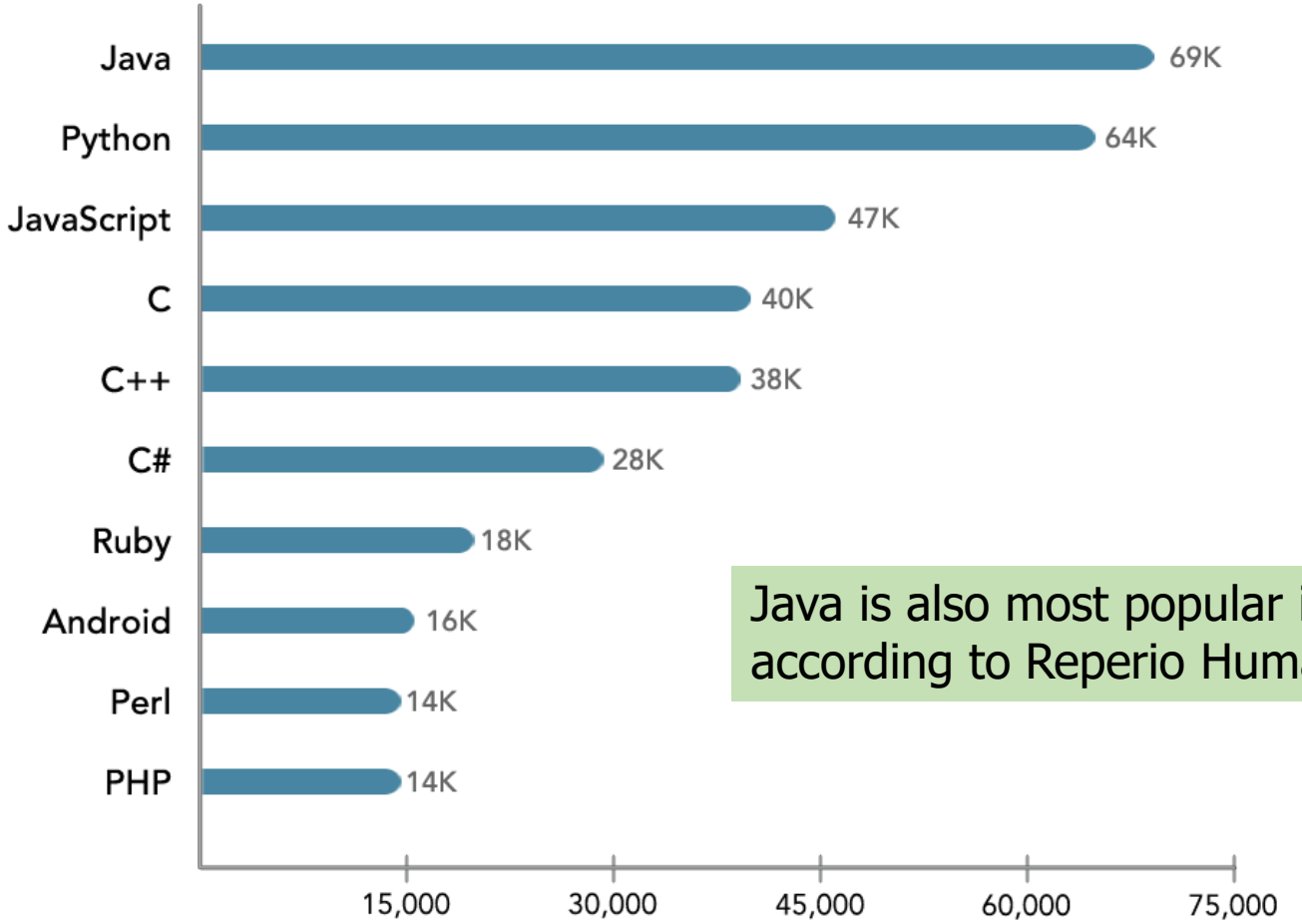
- **Programming Language & Java**
- Variables & Data Types:
  - ints, doubles
- Variable Operators:
  - addition, subtraction
- Selection:
  - if, else
- Iteration:
  - for, while, do

Programming language

- We will need to use some programming language to represent data structures and algorithms
- In this class, we will use the Java language
- However, you could use any other programming language to encode the same ideas - another popular language is C++

Most in-demand programming languages of 2019

Based on Indeed.com job postings in the USA - Feb 1, 2019



Java is also most popular in **Ireland** according to Reperio Human Capital [1]

Image Source: CodingNomads

[1]. https://www.reperiohumancapital.com/news/programming-languages-in-demand-ireland

IEEE Spectrum / Top Programming Languages

Language Ranking: IEEE Spectrum

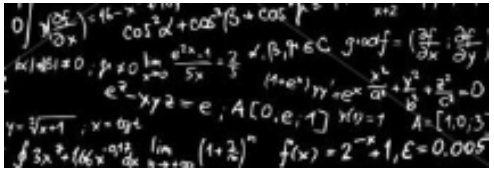
Rank	Language	Type	Score
1	Python		100.0
2	Java		95.4
3	C		94.7
4	C++		92.4
5	JavaScript		88.1
6	C#		82.4
7	R		81.7
8	Go		77.7
9	HTML		75.4

Source: https://spectrum.ieee.org/top-programming-languages/#toggle-gdpr

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#

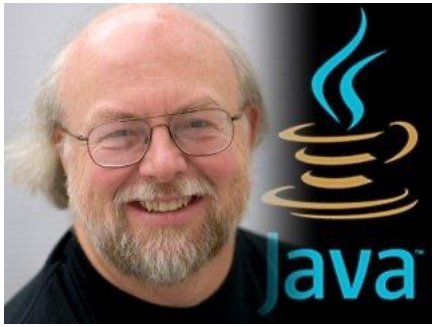
C++
C

Assembly language

Electronic circuits

Java programming

- Java is a programming language first released in 1995 originally developed by James Gosling at Sun Microsystems

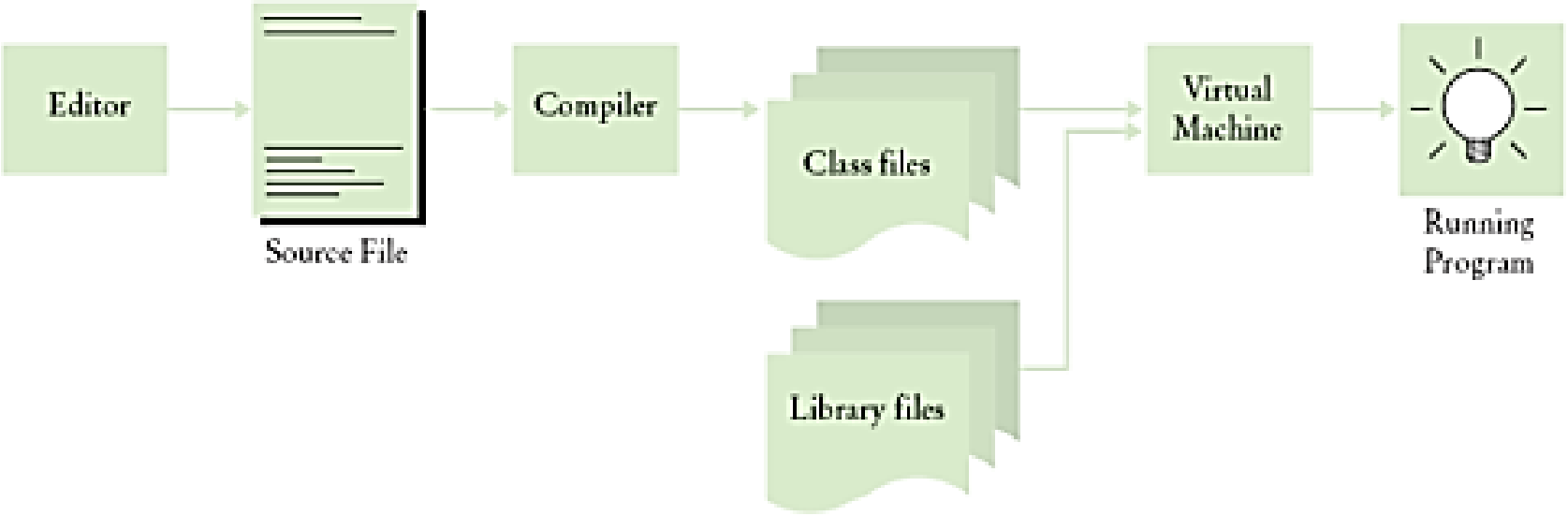


- One reason Java is popular is because it is **platform independent**
- Programs written in Java can run on any hardware or operating-system
- Compiled code is run on a Java Virtual Machine (JVM) which converts it to the native language

Platform independence

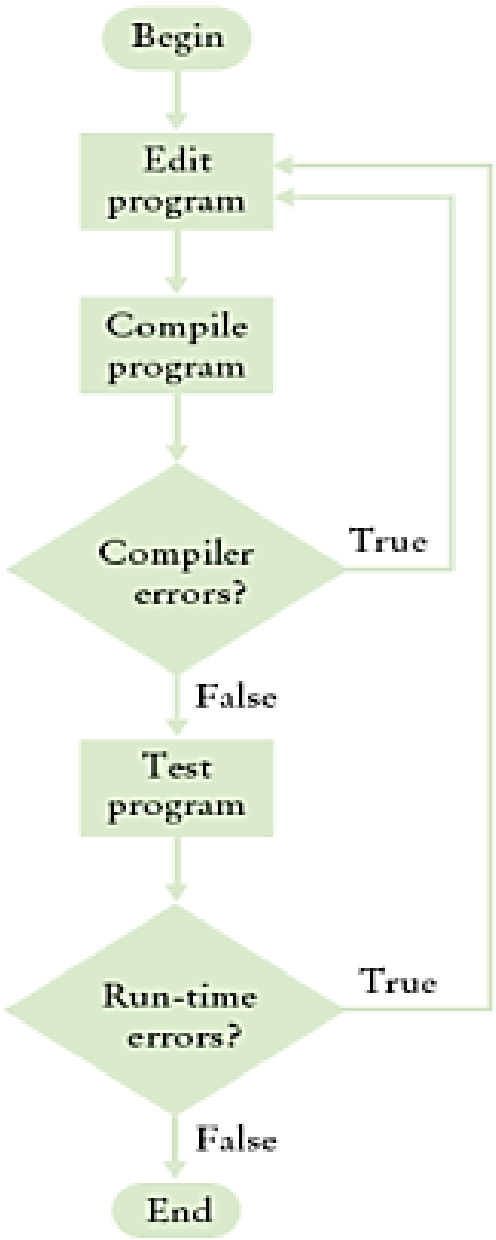
- Turing showed that machine, software and input can all be represented in terms of patterns of information
- The compiler translates the Java code into machine code that the JVM can run
  - JVM: Java Virtual Machine
- The JVM is a machine simulated by the actual physical machine it is running on

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



Outline

- Programming Language & Java
- **Variables & Data Types:**
  - int, double
- 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++){
        ...
}
```

// The life time of variable 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;

assignment

- 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

Assignment

- You can assign only a value to a variable that is consistent with the variable's declared type

```
int total;
int exam;
total = 55;    // legal
                // assignment
exam = 90.5  // illegal: 90.5 is not an integer.
```

Primitive types

- There are exactly eight primitive data types in Java
- Four of them represent integers:
  - byte, short, int, long
    - Example. 1, 11, 65536, -99
- Two of them represent floating point numbers:
  - float, double
    - Example. 3.1416, 0.33
- One of them represents characters:
  - char
    - Example. ‘s’, ‘A’
- 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


```
int i, j;
i = -2147483648;
j = i-1;
System.out.println(j);
```

A: Error

B: -2147483649

C: 2147483647

- Q: What is the appropriate data type for storing the number of Facebook users?


```
A: byte
B: short
C: int
D: long
```

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
```

- Example.
  - long rounded = Math.round(-3.7);
  - long rounded = Math.round(-4.2);

https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html

Examples of Math.round()

- Math.round() converts a floating-point number to **nearest** integer

Math.round(11.5) = 12
Math.round(-11.5) = -11

Math.round(11.46) = 11
Math.round(-11.46) = -11

Math.round(11.68) = 12
Math.round(-11.68) = -12

Outline

- Programming Language & Java
- Variables & Data Types:
  - int, double
- **Variable Operators:**
  - addition, subtraction
- Selection:
  - if, else
- Iteration:
  - for, while, do

Arithmetic expressions

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

	operator
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**

- Example. (In Java)
  - $55 \% 10 = 5$
  - $42 \% 4 = 2$
  - $10 \% (-3) = 1$
  - $\mathbf{(-10)} \% 3 = -1$
  - $-10 \% 3 = -1$
  - $\mathbf{(-10)} \% (-3) = -1$
  - Check those by yourself.

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

A: (number - (number % 7))/ 7
B: number / 7 – ((number / 7) % 1)
C: number / 7
D: All
E: A and B

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 (BODMAS 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;

count++; ⇔ count = count + 1;

count--; ⇔ 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;
```

```
num += count;  ←→  num = num + count;
num -= count;  ←→  num = num - count;
```

- Compare the different between
  - i++ and ++i
  - i-- and --i

- If we set
  - int x;
  - int i = 5;
- When we run the following codes, please show the value of x.
  - Code 1: x = i++;
  - Code 2: x = ++i;
  - Code 3: x = i++;
    x = ++i;

A: 5,6,7

B: 6,6,7

C: 5,6,6

Relational operators

operator		Example. a = 5, b = 5, c = 6
>	greater than	a > b // false c > b // true
>=	greater than or equal to	a >= b // true b >= c // false
<	less than	a < b // false b < c // true
<=	less than or equal to	a <= b // true c <= b // false

==	equal to	a == b // true
		b == c // false

!=	not equal to	a != b // false b != c // true

Frequent mistake!!

- If we want to put the variable “number” equal to 10 we use one “equal” sign

```
number = 10;
```

- However, if we want to check *if* number is equal to 10 then we use a double equals

```
if (number == 10)  //OK
```

- *What happen if we write “if (number = 10)”?*

The Math class

- Math class: contains methods for performing basic numeric operations like sqrt (square root) and pow (power of)
- 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)
  - **Math.sqrt(x): $\sqrt{x}$**

The Math class

- In Java,

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

can be represented as

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

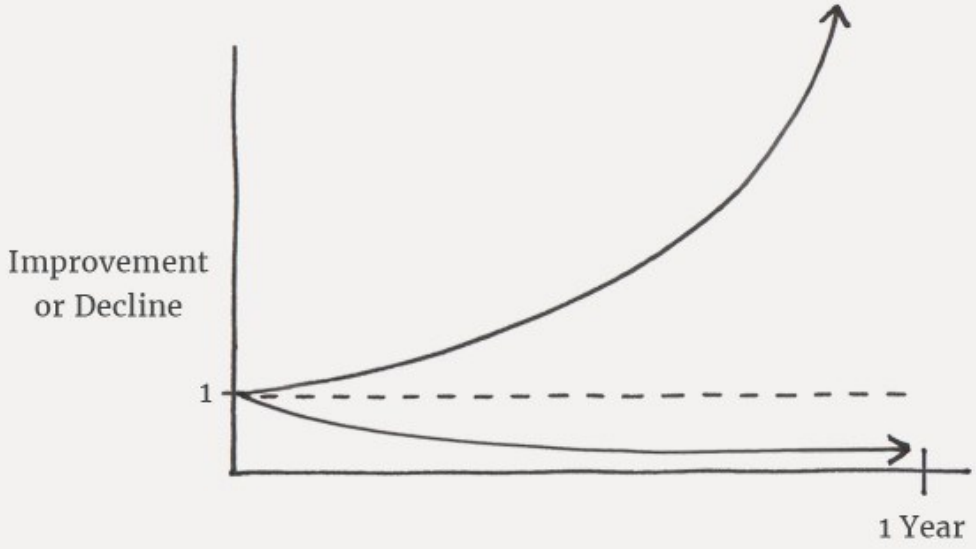
Mathematical methods in Java

Function	**Result**
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

The Power of Tiny Gains

1% better every day $1.01^{365} = 37.78$

1% worse every day $0.99^{365} = 0.03$



JamesClear.com

Mathematical methods in Java

Function	Example
Math.sqrt(x)	Math.sqrt(100) ←→ 10
Math.pow(x, y)	Math.pow(10, 3) ←→ 1000
Math.exp(x)	Math.exp(3) ←→ e^3

Math.log(x)	Math.log(-1); // NaN, out of range
	Math.log(0); // -Infinity
	Math.log(1); // 0
	Math.log(10); // 2.302585092994046

Math.sin(x), Math.cos(x), Math.tan(x)	Math.sin(0); // 0
	Math.sin(1); // 0.8414709848078965
	Math.sin(Math.PI / 2); // 1

Math.round(x)	Math.round(20.3) ←→ 20 Math.round(-20.5) ←→ -20 Math.round(-20.51) ←→ -21
Math.min(x, y) Math.max(x, y)	Math.min(5, 3) ←→ 3 Math.max(5, 3) ←→ 5

Questions

- What is the value of 643 / 100?
- What is the value of 643 / 100.0?
- What is the value of 643 % 100?
- What is the value of 643 % 100.0?

A: 6, 6.43, 43, 43.0
B: 6.43, 6.43, 43.0, 43.0
C: 6, 6.43, 43, 43

- Why doesn't the following statement compute the average of s1, s2, and s3?

```
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:

```
""
```

  - Example.

```
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
```

- What happens if we change n to double type?

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);

  - Example.

    a = Integer.parseInt(“123”); ←→ a = 123;

- Convert to string:

  String str = "" + n;

  str = Integer.toString(n);

  - Example.

    a = **Integer.toString**(123); ←→ a = “123”;

Substrings

- String greeting = "Hello, World!";
- String sub = greeting.substring(0, 5); // sub is "Hello”
  - Supply start and stopping index
  - First position is at 0

stop

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

greeting.substring(0, 5)

Substrings

- String greeting = "Hello, World!";
- Syntax is (**starting index, stopping index**)

  greeting.substring(start index, stopping index)
- Stops before it gets to the stopping index
- Substring length is ‘stopping index – starting index’

greeting.substring(7, 12)

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

stop

greeting.substring(7, 12)

Questions

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

2. Assuming the String variable college holds the value "Maynooth", what is the value of **college.substring(1, 2)**?
   - college = "Maynooth”;

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

Answer of Question 1

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

  s = “Hello”;

- We have that s.length() is 5.
- Then s + s.length() is “Hello5”.
- That is

  s is set to the string “Hello5”

Answer of Question 2

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

college = "Maynooth";

- We have that

college.substring(1, 2) is the string “a”

Answer of Question 3

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

  college = "Maynooth";

- We have that college.length() is 8.
- Then college.substring(2, college.length() - 3) is

  college.substring(2, 5)

- Thus,

  college.substring(2, 5) is set to 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

Is s.charAt(n) equal to s.substring(n, n+1)?

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

What is the output?


```
String s1 = "CS210";
String s2 = "CS210";
String s3 = new String("CS210");
String s4 = new String("CS210").intern();
if (s1 == s2)
        System.out.print("1");
if (s1 == s3)
        System.out.print("2");
if (s1 == s4)
        System.out.print("3");
if (s1.equals(s2))
        System.out.print("4");
if (s1.equals(s3))
        System.out.print("5");
if (s1.equals(s4))
        System.out.print("6");
```

A: 123456

B: 13456

C: 456

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

More String comparisons

```
public class MyTest {
    public static void main(String[] args) {
        String str1 = "abc";
        String str2 = "IloveU";
        System.out.println(str1.compareTo(str2));
    }
}
```

- Output: 24
  - The ASCII code of ‘a’ is 97 and the ASCII code of ‘I’ is 73.
  - So str1.compareTo(str2) return 97 - 73 = 24.

More String comparisons

```
public class MyTest {
    public static void main(String[] args) {
        String str1 = "aIloveU520";
        String str2 = "abc";
        System.out.println(str1.compareTo(str2));
    }
}
```

- Output: -25
  - The ASCII code of ‘I’ is 73 and the ASCII code of ‘b’ is 98.
  - So str1.compareTo(str2) return 73 - 98 = -25.

More String comparisons

```
public class MyTest {
    public static void main(String[] args) {
        String str1 = "ljj";
        String str2 = "ljjLovexql";
        System.out.println(str1.compareTo(str2));
    }
}
```

- Output: -7
  - string str1 == string str2.substring(0, str1.length())
  - So str1.compareTo(str2) return the difference of their length
    - 3 - 10 = -7.

More String comparisons

- s1.compareTo(s2) < 0
  - the string s1 comes before the string s2 in the **dictionary**
    - Example. s1 = ‘abc’ and s2 = ‘af’

- s1.compareTo(s2) **== 0**
  - the string s1 is equal to the string s2

- s1.compareTo(s2) **> 0**
  - the string s1 comes after the string s2 in the **dictionary**
    - Example. s1 = ‘abc’ and s2 = ‘aac’

Reading input

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

- **nextInt** reads an integer value
- **nextDouble** reads a double value
- **nextLine** reads a line (until user hits **ENTER**)
- **next** reads a word (until any **white space**)

- You will need to include this line at the top:

import java.util.Scanner;

Reading input

```
public class Test {
        public static void main(String[] args) {

                Scanner sc = new Scanner(System.in);
                System.out.println("Scanner test. Input a string: ");
                while (true){
                        String line = sc.nextLine();   // input
                        if (line.equals("exit")) break;   // this code will end at here
                        System.out.println(">>> "+line);   // output our input
                }

        }
}
```

Result:

```
============================
Scanner test. Input a string:
234
>>> 234
a test code
>>> a test code
exit
```

Outline

- Programming Language & Java
- Variables & Data Types:
  - int, double
- Variable Operators:
  - addition, subtraction
- Iteration
- **Selection:**
  - if, else
- Iteration:
  - for, while, do

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**
    - **;**
    - Example.
      - int i;
      - i – 10;
  - Selection: if / else
  - Iteration: for/ while/ do
    - loops

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*
    - ***If-else if-else***

- Conditional test is an expression that results in a boolean value using relational operators
- If we have the statement **int x = 3**, then the conditional test **(x >= 2)** evaluates to true

The if Statement

- The *if statement* has the following syntax:

```
if (condition)
{
    statement;
}
```

- **if** is a Java reserved word
- The ***condition*** must be a boolean expression.
  - That is **its output is true or false**.
- It must evaluate to either true or false.
- If the ***condition*** is true, the ***statement*** is executed. If it is false, the ***statement*** is skipped.

The if Statement

```
if (condition)
{
    statement1;
    statement2;
    …
}
statement3;
```

// Condition is true:
// Do statement1, statement2, and keep on

// condition is false:
// Skip statement1, statement2, …
// Go to statement 3.

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**

The if-else Statement

```
public class Test {
        public static void main(String[] args) {
                int score = 30;
                if( score >= 60 ){
                        System.out.print("You passed this course");
                } else{
                        System.out.print("You didn't pass this course ");
                }
        }
}
```

Result:

==========

You didn’t pass this course

The if-else if-else Statement

```
if ( condition1 )
    statement1;
else if ( condition2 )
    statement2;
else
    statement3;
```

- If ***condition1*** is **true**, *statement1* is executed
- If ***condition1*** is **false** and ***condition2*** is **true**, *statement2* is executed
- If both ***condition1*** and ***condition2*** are **false**, *statement3* is executed
- **Only one statement will be executed.**

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("Number of right guesses" +
    correct);
    correct++;
}
```

// block statement 1

```
else {
    System.out.println("Number of wrong guesses" +
    wrong);
    wrong++;
}
```

// block statement 2

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

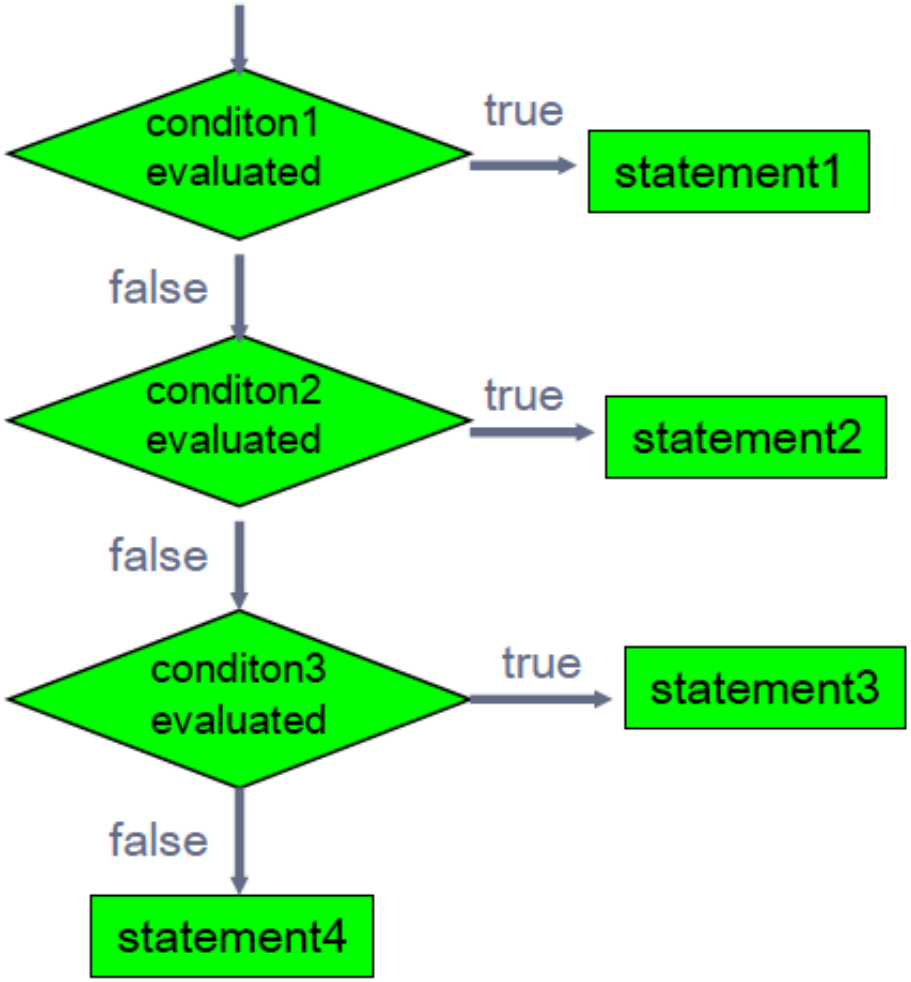
Nested if example

```
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 numGrade = 83.6;
char letterGrade;
if (numberGrade >= 89.5) {
    letterGrade = 'A';
} else if (numGrade >= 79.5) {
    letterGrade = 'B';
} else if (numberGrade >= 69.5) {
    letterGrade = 'C';
} else if (numGrade >= 59.5) {
    letterGrade = 'D';
} else {
    letterGrade = 'F';
}
System.out.println("My grade is " + numGrade + ", " + letterGrade);
```

Result:

==========

My grade is 83.6, **B**

Logical operators

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

Symbol	Logical
!	NOT
&&	AND
||	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

- 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…");
```

  - Condition 1: **((total < MAX+5)**
  - Condition 2: **!found**
  - Here, if we want to do System.out.println ("Processing…"), we need both condition 1 and condition 2 holds
- 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

Outline

- Programming Language & Java
- Variables & Data Types:
  - int, double
- Variable Operators:
  - addition, subtraction
- Selection:
  - if, else
- **Iteration:**
  - for, while, do

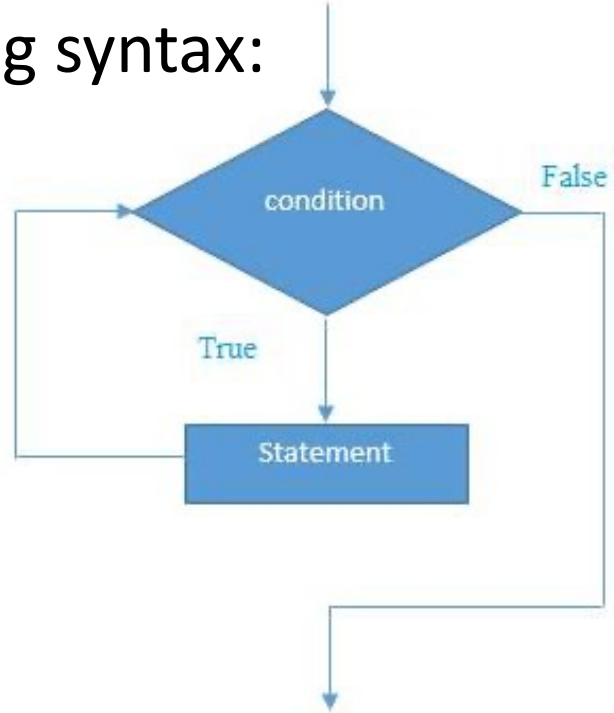
Iteration

- ***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:

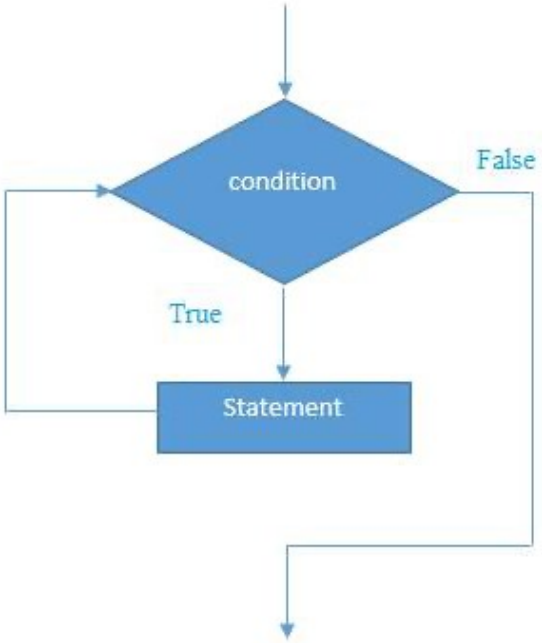
while (*condition*)
** *statement*;**



- **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

- Example 1.

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

- Example 2.

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

Result of Example 1:

==========

1
2
3
4
5

Result of Example 2:

==========

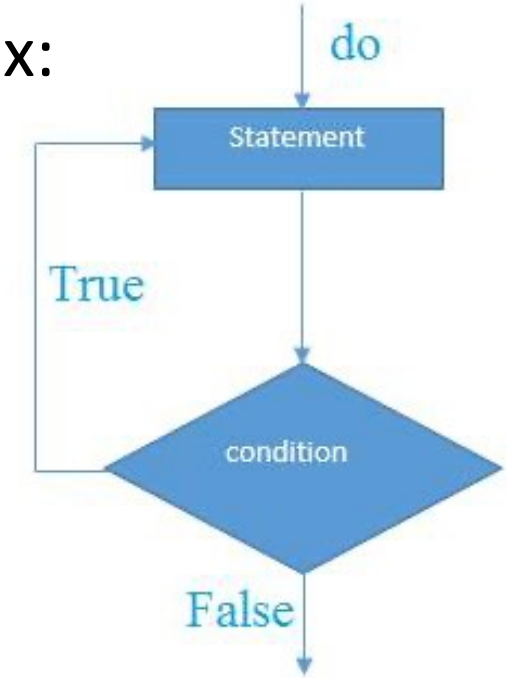
1
2
3
4
5
6
7
8
9
…
…

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

- **do and while are** reserved words
- 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

- Example 1.

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

- Example 2.

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

Result of Example 1:

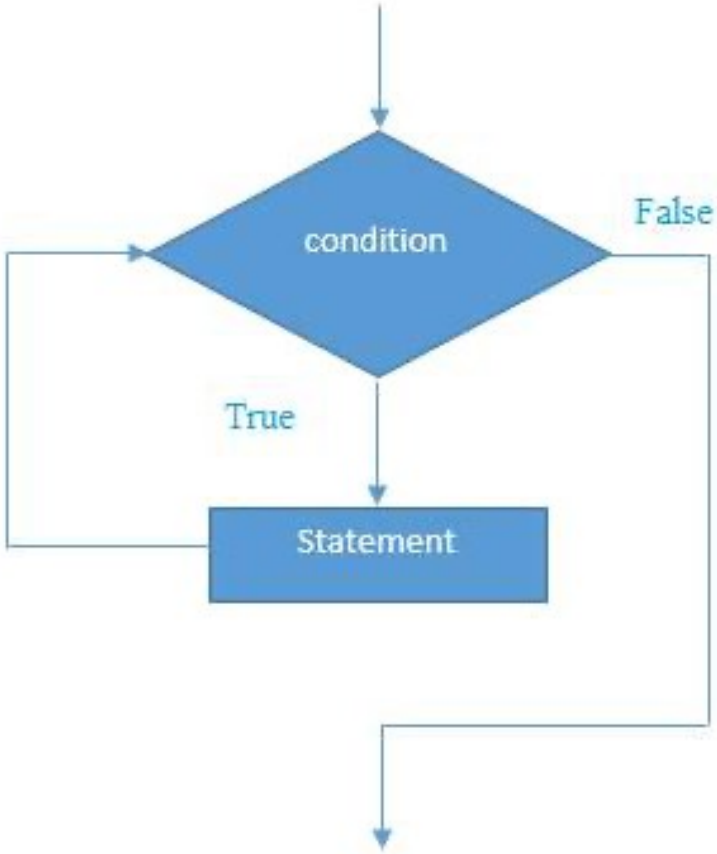
```
==========
1
2
3
4
5
```

Result of Example 2:

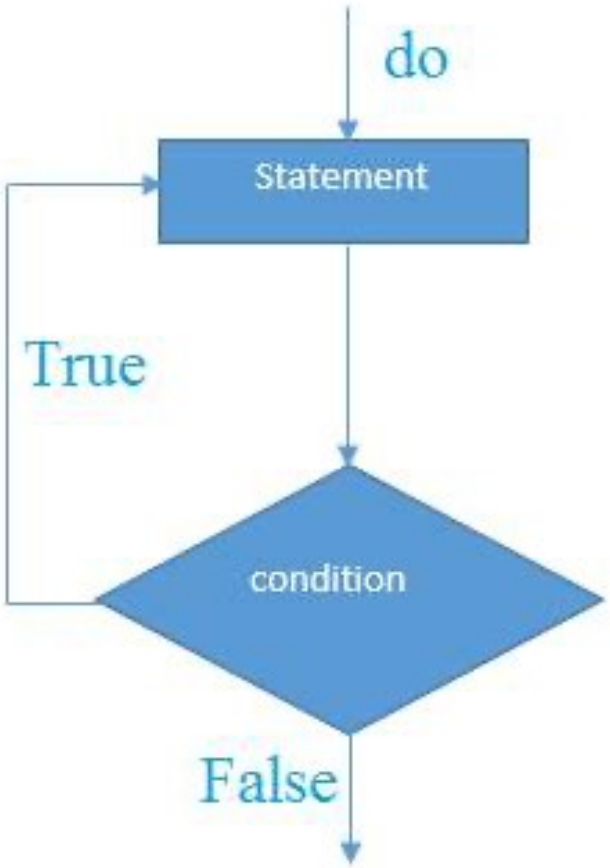
```
==========
1
```

Comparing while and do

while loop



do loop



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 loop 1
    do {                              // do loop 2
    } while (...);
while (...);
```

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

The for Statement

- The *for statement* has the following syntax:

for (*initialization*; *condition*; *increment*)
statement;

- **for** is reserved word
- 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 ***condition-statement-increment*** cycle is executed repeatedly

Example

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

Result

==========

hello: 0

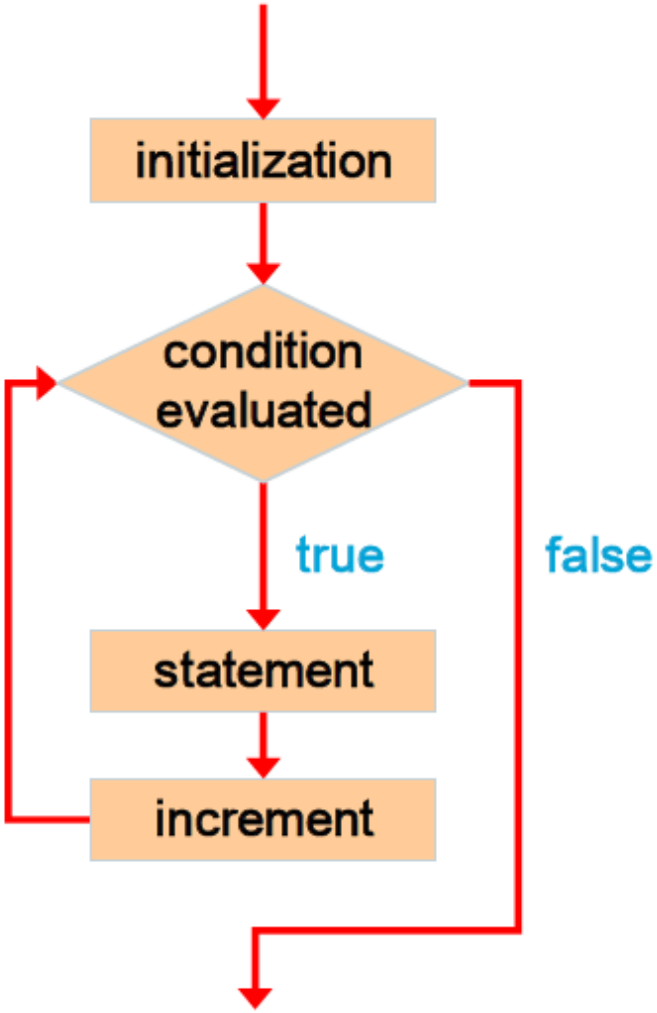
hello: 1

hello: 2

hello: 3

hello: 4

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
  - Example.

```
for (int i = 0; i < -1; i++) {
    System.out.println("hello: " + i);
}
```

- 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);
}
```

Result

==========

1

2

3

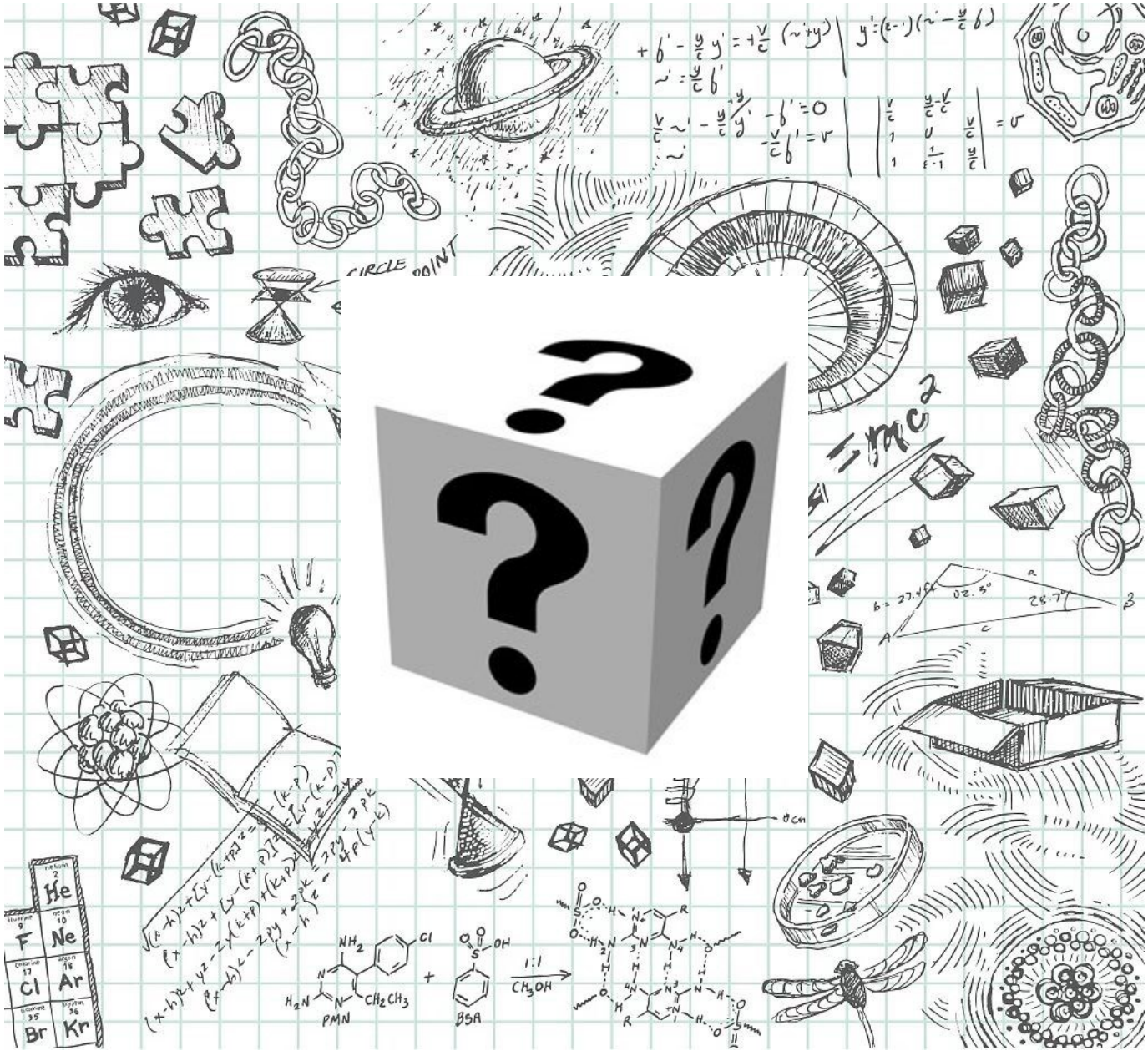
4

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


CIRCLE POINT
PMN
BSA
1:1
CH3OH