

Quick review – Giving an output

```
// Give some output:  
System.out.println("In this computer light travels "  
+ number*1000*299792.458 + " meters per second.");
```



Quick review – Switching values

```
int tmp1 = a;  
int tmp2 = b;  
a = tmp2;  
b = tmp1;
```

```
int tmp = a;  
a = b;  
b = tmp;
```



Quick review – Basic structure

```
public class MyProgram {  
  
    public static void method() {  
        // Body of the method  
  
    }  
  
    public static void main(String[] args) {  
        // Body of the main method  
  
    }  
}
```



Quick review – Methods

Methods are named modules of code. Their syntax is:

```
<scope-indicator> <return-indicator> <name>(<parameter list>)  
{ <body of the method>}
```

where

<scope-indicator> can be private, public, static, etc.;

<return-indicator> can be some data type or void.

<name> is the name of the method. It can be any valid Java identifier.

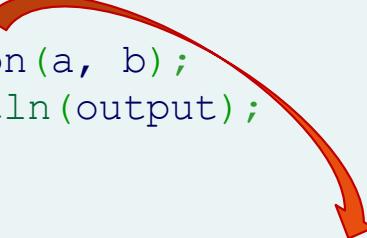
<parameter list> is a list of declarations of variables used;

<body of the method> is a list of valid Java statements. **If *<return-indicator>* is some data type, then *<body of the method>* has to contain return statement!**



Quick review – Example: Method

```
public static void main(String[] args) {  
    double output;  
    int a = 12;  
    int b = 3;  
    output = division(a, b);  
    System.out.println(output);  
}  
  
public static double division (int dividend, int divisor)  
{  
    double result;  
    result = dividend/divisor;  
    return result;  
}
```



Quick review – Example: Variable scope

- We can use the same variable name with different scopes depending on where it is declared

```
public class Main {  
    static int a; // Global variable: Class level scope  
  
    public static void myMethod() {  
        int a; // Local variable: Method (myMethod) level scope  
    }  
  
    public static void main(String[] args) {  
        int a; // Local variable: Method (main) level scope  
    }  
}
```



Introduction to Computer Science I

Module 3 – Decisions... decisions



Maastricht University

| Department of Advanced Computing Sciences

Overview

Control flow

Flowcharts

if, if-else, if-else-if

Conditions

- Boolean expressions
- Lazy evaluation

Switch-case

Enumerated Types

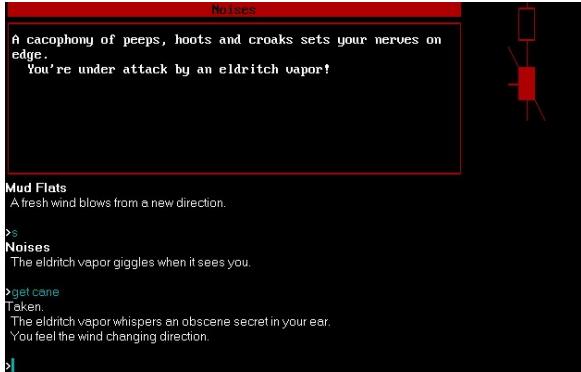
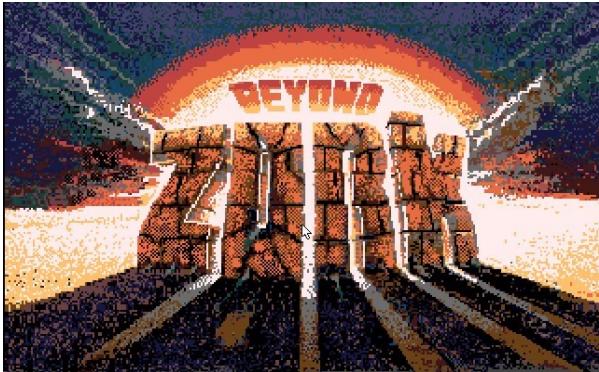
Tracing and logging

BONUS TRACKS:

- More on Scanner
- More on String

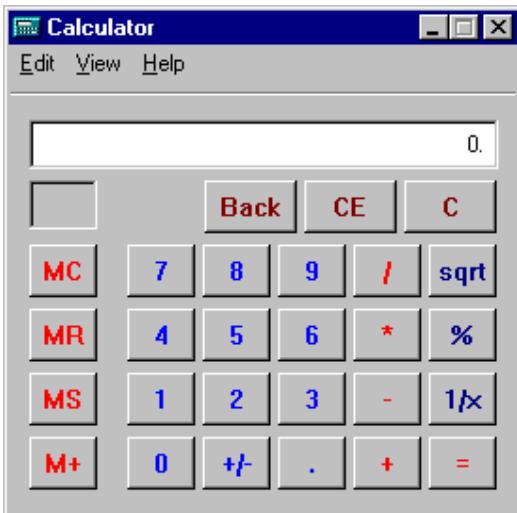


Control Flow



Control Flow

Your amazing DIY software
to perform
basic mathematical operations



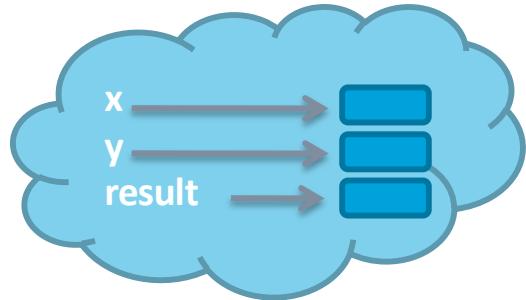
Pseudo-code!

```
if key_pressed is "+" then
    result = a + b
else if key_pressed is "-" then
    result = a - b
...

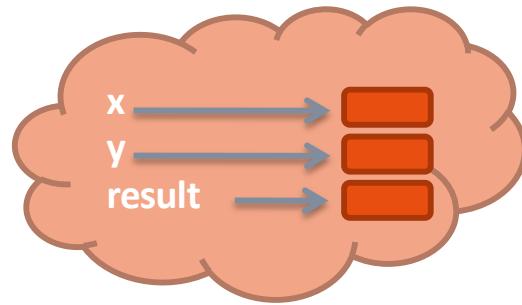
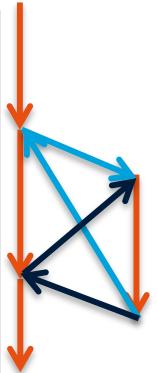
```



Control Flow – Refreshing your knowledge



```
public static void main(String[] args) {  
    double x = 5.0;  
    double y = 12.3;  
    double result = function(x,y);  
    System.out.println("The functionvalue at");  
    System.out.println("the point (" + x + "," + y + ")");  
    System.out.println("is " + result);  
  
    x = -15;  
    y = 2.34;  
    result = function(x,y);  
    System.out.println("The functionvalue at");  
    System.out.println("the point (" + x + "," + y + ")");  
    System.out.println("is " + result);  
  
}  
  
public static double function(double x, double y) {  
    double result;  
    result = Math.exp(x) + y*y;  
    return result;  
}
```



Flowcharts

- Generic steps



- Input/Output



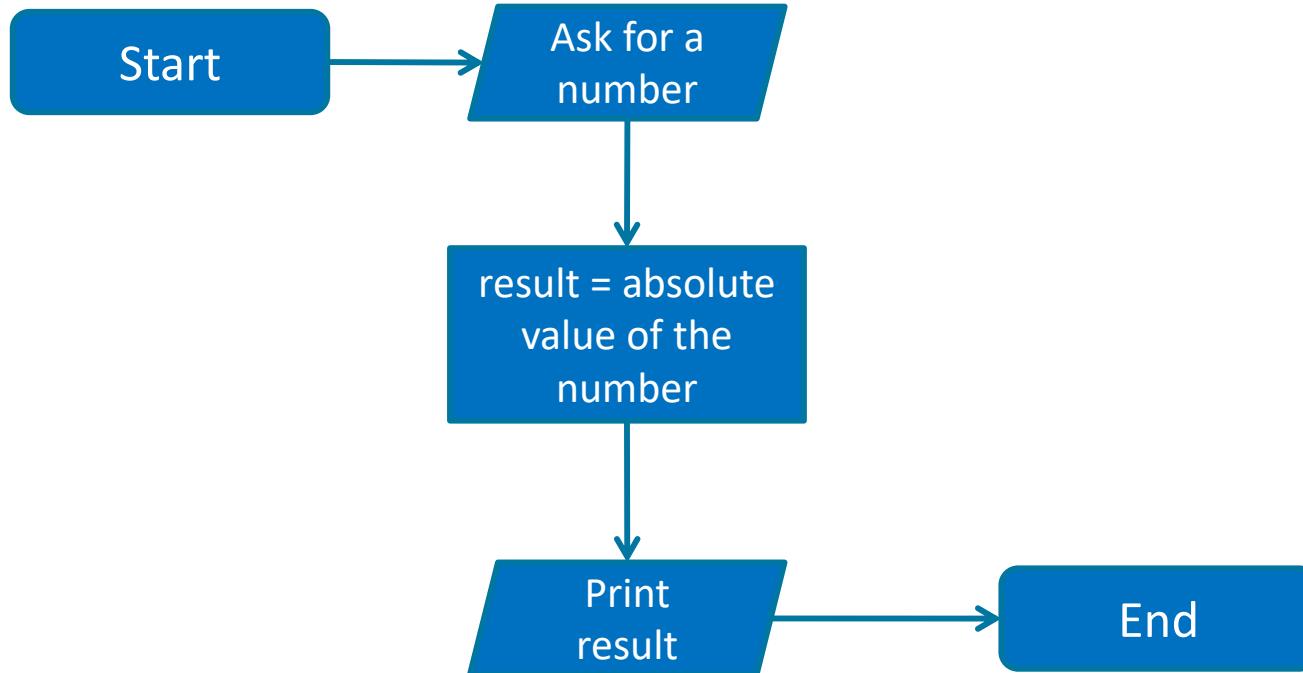
- Arrows: control flow



- (Begin and end symbols)



Example flowchart



Flowcharts

- Generic steps



- Input/Output



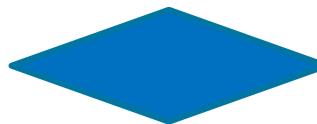
- Arrows: control flow



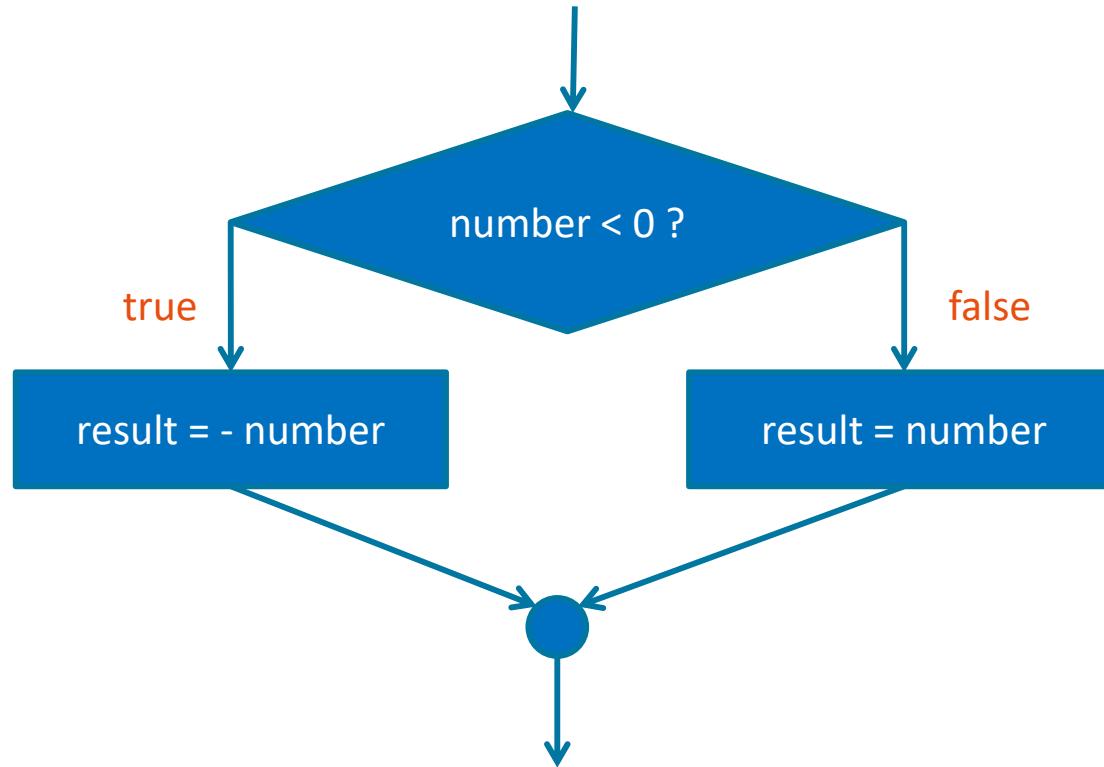
- (Begin and end symbols)



- Condition

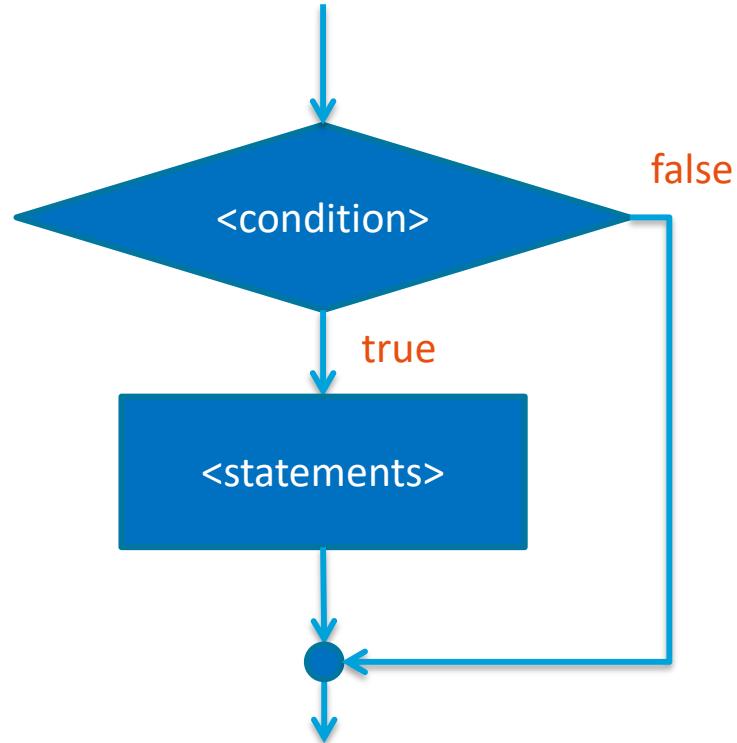


Conditional control flow



IF

Conditional execution of statements

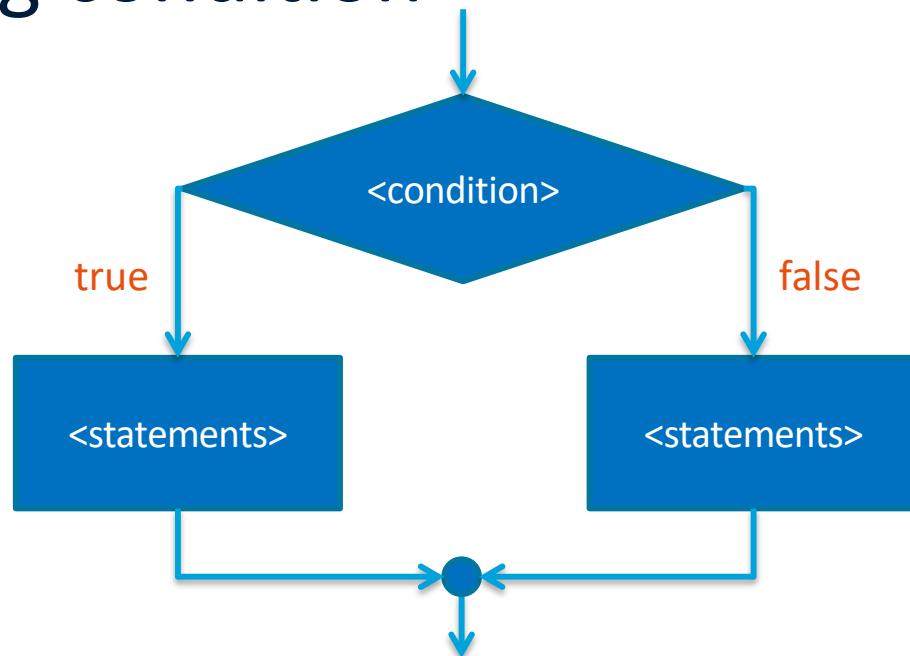


```
if (<condition>) {  
    <statements>  
}
```

```
if (password == "Help1234") {  
    access_granted = true;  
}
```

IF – ELSE

When there is also something to do under a failing condition



```
if (<condition>) {  
    <statements>  
} else {  
    <statements>  
}
```

Conditions

Boolean expressions

- Evaluate **true** or **false**

1. Boolean variables

2. Relational operators

`== != < <= > >=`

3. Boolean operators

`&& || !`

4. Methods that return a Boolean value

E.g. `(a != 4) && (b-5.6 <= a)`

Operator Precedence in Java:

<https://introcs.cs.princeton.edu/java/11precedence/>

- Evaluation from left to right
- Importance of parenthesis



Example code

```
public static void main(String[] args) {  
  
    Scanner in = new Scanner(System.in);  
    double number = in.nextDouble();  
    double result = 0.0;  
  
    if (number >= 0){  
        result = number;  
    } else {  
        result = -number;  
    }  
    System.out.println(result);  
}
```

```
public static void main(String[] args) {  
  
    Scanner in = new Scanner(System.in);  
    double number = in.nextDouble();  
    double result = 0.0;  
  
    if (number >= 0)  
        result = number;  
    else  
        result = -number;  
  
    System.out.println(result);  
}
```

```
public static void main(String[] args) {  
  
    Scanner in = new Scanner(System.in);  
  
    double result = in.nextDouble();  
  
    if (result < 0)  
        result = - result;  
  
    System.out.println(result);  
}
```

Floating point comparisons

Be careful with floating point precision!

```
double a = 2.0;
double b = Math.sqrt(a);

if (a == b*b) {
    System.out.println("Equal!");
} else {
    System.out.println("Not Equal!");
    System.out.println(a + " != " + b*b);
}
```

```
wopr: java$ java Main
Not Equal!
2.0 != 2.0000000000000004
wopr: java$ █
```

Floating point comparisons (2)

Solution: allow some error for doubles, use for example
1E-14

```
final double EPSILON = 1E-14;

double a = 2.0;
double b = Math.sqrt(a);

if (Math.abs(a - b*b) <= EPSILON) {
    System.out.println("Equal!");
} else {
    System.out.println("Not Equal!");
}
```

```
wopr: java$ java Main
Equal!
wopr: java$ █
```

String comparisons

Strings are not a basic type!

Do NOT use `==` to compare String type variables

Use the method “equals”:

```
String a = "Joe";
String b = "J";
b = b+"oe";
if (a == b) System.out.println("== works fine.");
else if (a.equals(b)) System.out.println("Better use equals");
else System.out.println("Typo?");
```

```
wopr: java$ java Main
Better use equals
wopr: java$
```

Pay attention to
method call!!!!

String comparisons

```
String a = "Joe";  
  
if (a == a) System.out.println("== works fine.");  
else if (a.equals(a)) System.out.println("Better use  
equals");  
else System.out.println("Typo?");
```

But... wait. Why is this statement returning “***== works fine.***”?



Dolian
McMitchell



DACS
student = "Dolian"

Yes sir, Dolion works
here

student == employee
False

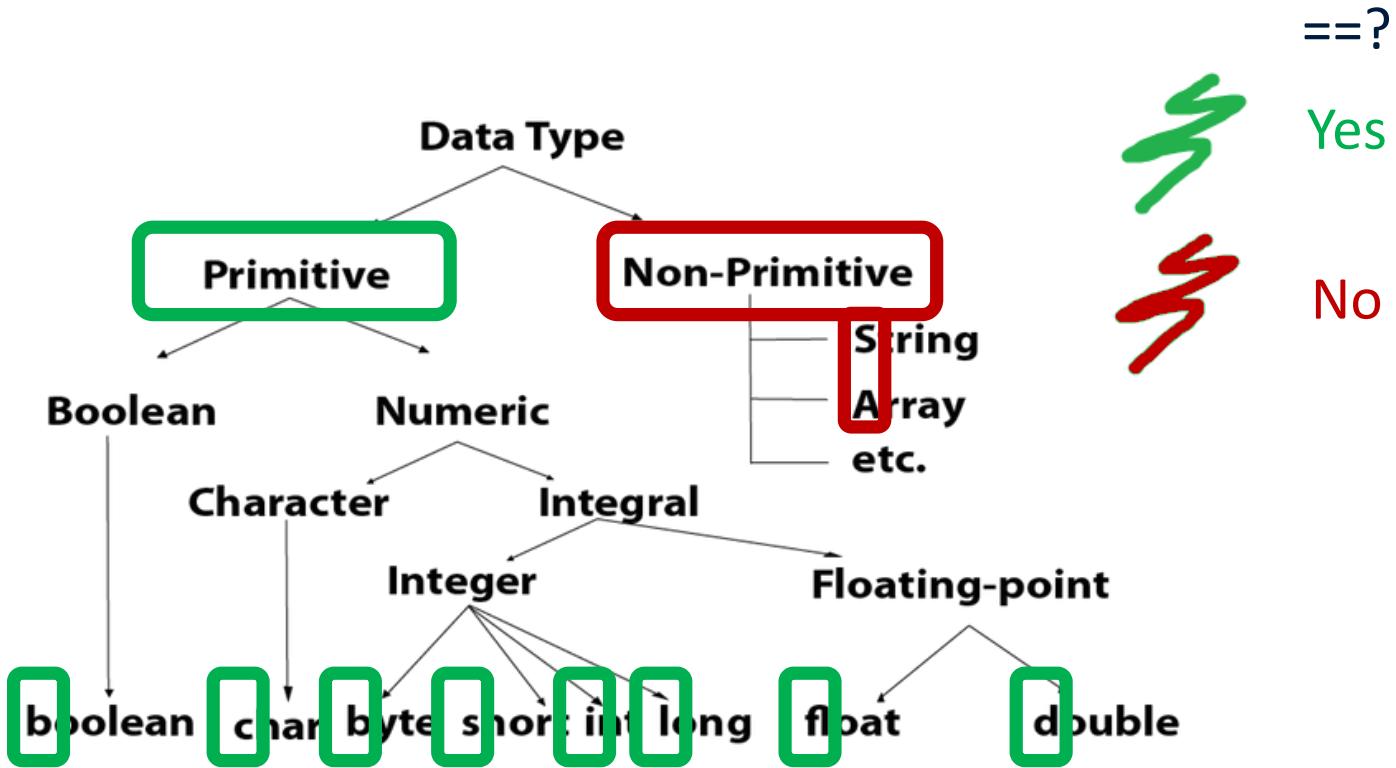
Someone named
Dolian studies at DACS



MASA
employee = "Dolian"



Mnemonic trick



<https://www.javatpoint.com/java-data-types>

More String comparisons

...

if (a.equals(b))

- Checks equality

if (a.equalsIgnoreCase(b))

- Checks equality but ignores case differences

if (a.compareTo(b) < 0)

➤ Lexicographic ordering

> 0

< 0 ➔ a comes before b

== 0

> 0 ➔ b comes before a

== 0 ➔ a.equals(b) == true

(all capital letters come before non-capitals)

Check homework/quiz



Math.random()

Generates a (pseudo-)random double in

[0.0, 1.0 [

Can be used to make your program act stochastically.

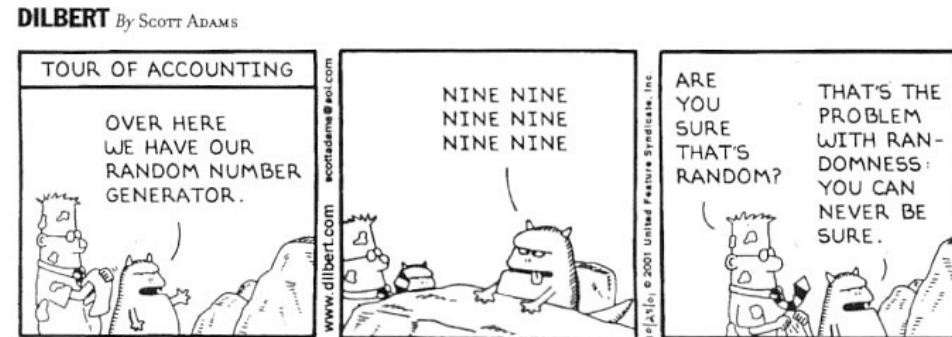
E.g. execute code with 50% probability

```
if (Math.random() < 0.5)  
    ...
```



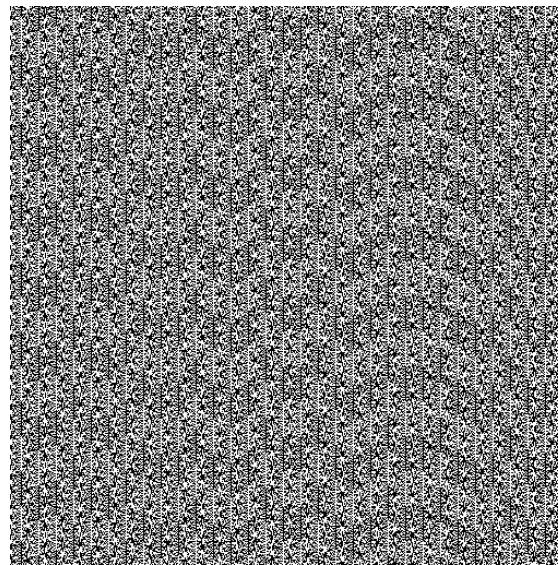
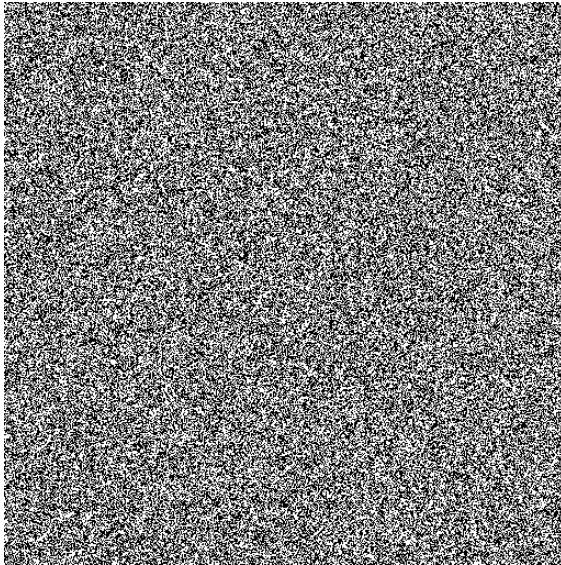
Why (pseudo-)random values?

- Computers are deterministic (its behaviour is completely predictable)
- They cannot generate random numbers
- Use of mathematical algorithms to produce “random enough” values
- Use of a seed as starting point (e.g. date)



Why (pseudo-)random values?

- Extreme example: Unfortunate combination of programming language, operating system, and function:



<https://boallen.com/random-numbers.html>

Multiple relational operators

```
if (0 < a < b)
```

Syntax error!

```
int a = 5;  
int b = 8;
```

```
int a = 5;  
int b = 3;
```

Why?

Trace the evaluation

1. $0 < a \Rightarrow$ resolves to boolean
2. $<\text{boolean}> < <\text{int}> \Rightarrow$ no such operator!

Combine multiple operators through **&&** and **||**

Instead:

```
if (0 < a && a < b)
```

or

```
if ((0 < a) && (a < b))
```

Boolean expression evaluation

... where being lazy can give you the best performance ...

```
double a = 2.0;
double b = Math.sqrt(a);

if a < 0.0 &&
   Math.abs(a - b*b) <= EPSILON &&
   b == Math.PI &&
   b <= a) {
```

Left to right evaluation;
stop as soon as possible!

```
if (a == 2.0 ||
    Math.abs(a - b*b) <= EPSILON ||
    b == Math.PI ||
    b <= a) {
```



Exploiting lazy evaluation

Division by 0

- runtime error that often can not be predicted

```
wopr: java$ java Division
Give dividend
13
Give divisor
0.0
Exception in thread "main" java.lang.ArithmetricException: / by zero
    at Division.integerDivision(Division.java:25)
    at Division.main(Division.java:13)
wopr: java$
```

Avoid by testing:

```
if (a!=0.0) {
    result = b/a;
    ...
}
```



Exploiting lazy evaluation (2)

What if condition on the result?

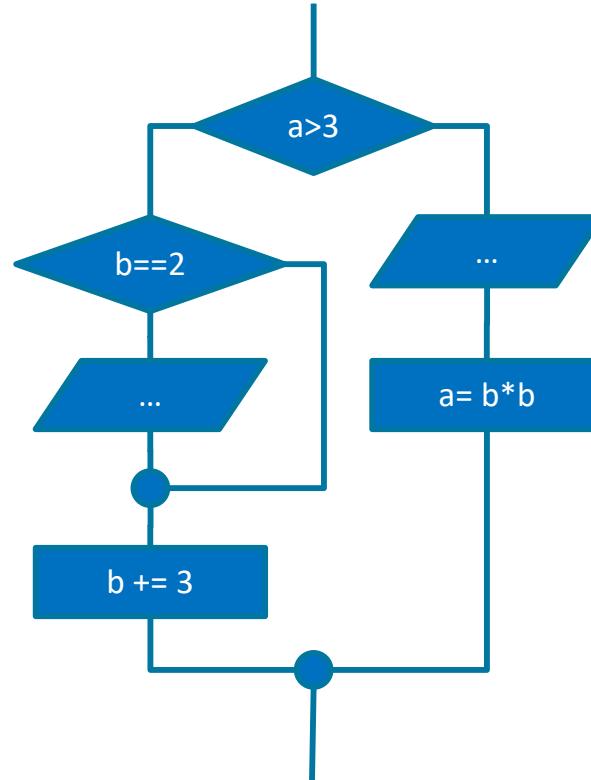
```
if (a!=0) {  
    if (b/a > 3) b/a;
```

```
if (a!=0.0 && b/a > 3) {  
    if == false then never executed
```

Nested conditionals

Conditional statements can appear as
statements of conditional statements
of conditional statements
of conditional statements
of conditional statements
of conditional statements
of conditional statements

```
if (a > 3) {  
    if (b == 2) {  
        System.out.println(" ... ");  
    }  
    b +=3;  
} else {  
    System.out.println(" ... ");  
    a = b*b;  
}
```



Special Case: Decision List

When dealing with number of mutually exclusive conditions:

```
if (<condition>) {  
    <statements>  
} else {  
    if (<condition>) {  
        <statements>  
    } else {  
        if (<condition>) {  
            <statements>  
        } else {  
            ...  
        }  
    }  
}
```



```
if (<condition>) {  
    <statements>  
} else if (<condition>) {  
    <statements>  
} else if (<condition>) {  
    <statements>  
} else if (<condition>) {  
    ...  
}
```

An example

```
Scanner in = new Scanner(System.in);
System.out.println("Please state your name:");
String name = in.nextLine();
System.out.println("Please state your role at UM");
String role = in.nextLine();

if (role.equals("student")) {
    System.out.println("Yo " + name + ", wanna grab a beer?");
} else if (role.equals("teaching assistant")) {
    System.out.println("Hello " + name + ", why aren't you working?");
} else if (role.equals("secretary")) {
    System.out.println("Goodmorning " + name + ". You look stunning today.");
} else if (role.equals("lecturer")) {
    System.out.println("Honorable dr. " + name + ", how can I serve you?");
} else {
    System.out.println("Euhm, hello?");
}
```



An example simplified

```
Scanner in = new Scanner(System.in);
System.out.println("Please state your name:");
String name = in.nextLine();
System.out.println("Please state your role at UM");
String role = in.nextLine();

if (role.equals("student"))
    System.out.println("Yo " + name + ", wanna grab a beer?");
else if (role.equals("teaching assistant"))
    System.out.println("Hello " + name + ", why aren't you working?");
else if (role.equals("secretary"))
    System.out.println("Goodmorning " + name + ". You look stunning today.");
else if (role.equals("lecturer"))
    System.out.println("Honorable dr. " + name + ", how can I serve you?");
else
    System.out.println("Euhm, hello?");
```



More simplification

```
Scanner in = new Scanner(System.in);
System.out.println("Please state your name:");
String name = in.nextLine();
System.out.println("Please state your role at UM");
String role = in.nextLine();

if (role.equals("student"))
    System.out.println("Hello " + name + ", wanna grab a beer?");
if (role.equals("teaching assistant"))
    System.out.println("Hello " + name + ", why aren't you working?");
if (role.equals("secretary"))
    System.out.println("Goodmorning " + name + ". You look stunning today.");
if (role.equals("lecturer"))
    System.out.println("Honorable dr. " + name + ", how can I serve you?");
else
    System.out.println("Euhm, hello?");
```



Switch – case

Special case decision list: Tests equality

```
System.out.println("Choose from menu:");
int choice = in.nextInt();
if (choice == 1)
    doMenu1();
else if (choice == 2)
    doMenu2();
else if (choice == 3)
    doMenu3();
else if (choice == 4)
    doMenu4();
else
    doTheDefault();
```

```
switch (choice) {
case 1: doMenu1(); break;
case 2: doMenu2(); break;
case 3: doMenu3(); break;
case 4: doMenu4(); break;
default: doTheDefault(); break;
}
```



- Can use
- byte, short, int, long and char
- Strings (since Java 7)
- Enumerated types
- Some wrapper classes

Enumerated Types

Since Java 5.0, you can declare your own enumerated types

```
public enum <typeName> {<value1>, ..., <valueN>}
```

```
public class Schedule {
    public enum WeekDay {Monday, Tuesday, Wednesday, Thursday,
Friday};

    public static void main(String[] args) {
// ...
        WeekDay today = WeekDay.Wednesday;
// ...
        System.out.println("Today is " + today);
        System.out.println("Today is " + today.name());
        System.out.println("Today is " + today.ordinal());
    }
}
```

```
Today is Wednesday
Today is Wednesday
Today is 2
```



Switch –case with enumerated type

```
enum Operation { PLUS, MINUS, TIMES, DIVIDE }

public static void main(String[] args) {
    int op1=10;
    int op2=2;
    Operation myOp = Operation.TIMES;

    System.out.println("The result of the " + myOp + " operation is ");

    switch(myOp) {
        case PLUS:      System.out.println(op1+op2); break;
        case MINUS:     System.out.println(op1-op2); break;
        case TIMES:     System.out.println(op1*op2); break;
        case DIVIDE:    System.out.println(op1/op2); break;
    }
}
```



On breaking and falling through

```
switch (choice) {  
    case 1: System.out.println("1"); break;  
    case 2: System.out.println("2"); break;  
    case 3: System.out.println("3"); break;  
    case 4: System.out.println("4"); break;  
    default: System.out.println("D"); break;  
}
```



(choice == 2) causes
“2” to be printed

```
switch (choice) {  
    case 1: System.out.println("1");  
    case 2: System.out.println("2");  
    case 3: System.out.println("3");  
    case 4: System.out.println("4");  
    default: System.out.println("D");  
}
```



(choice == 2) causes
“2”, “3”, “4” and “D” to be
printed

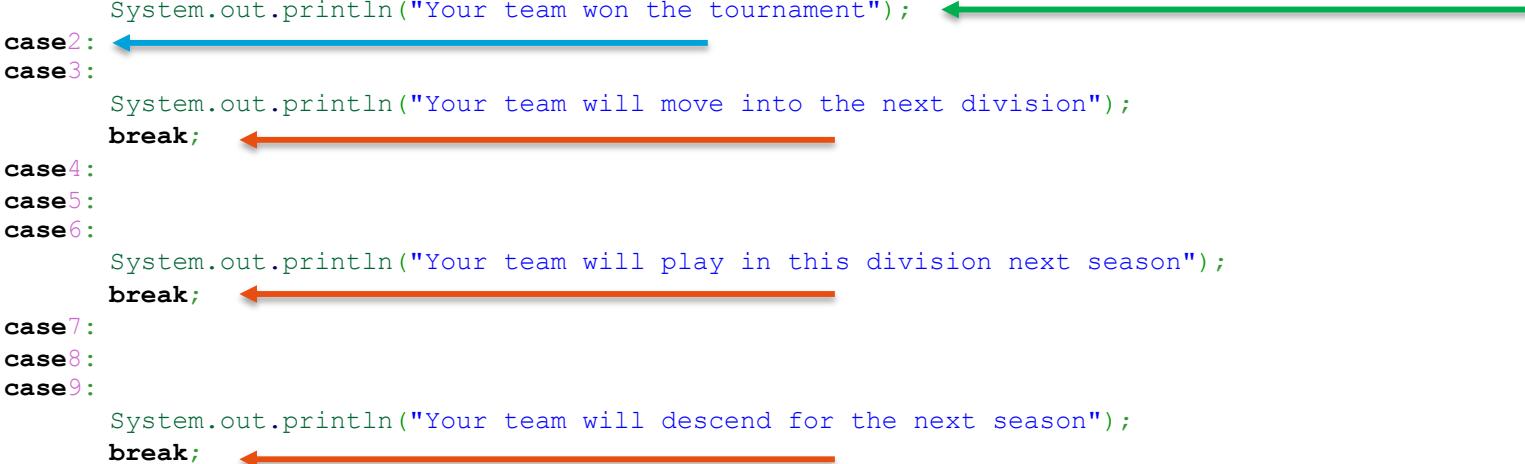
break; ends the executing of
the switch and jumps to end

Not using **break;** causes a
“fallthrough” where all code
is executed from the
successful case onwards

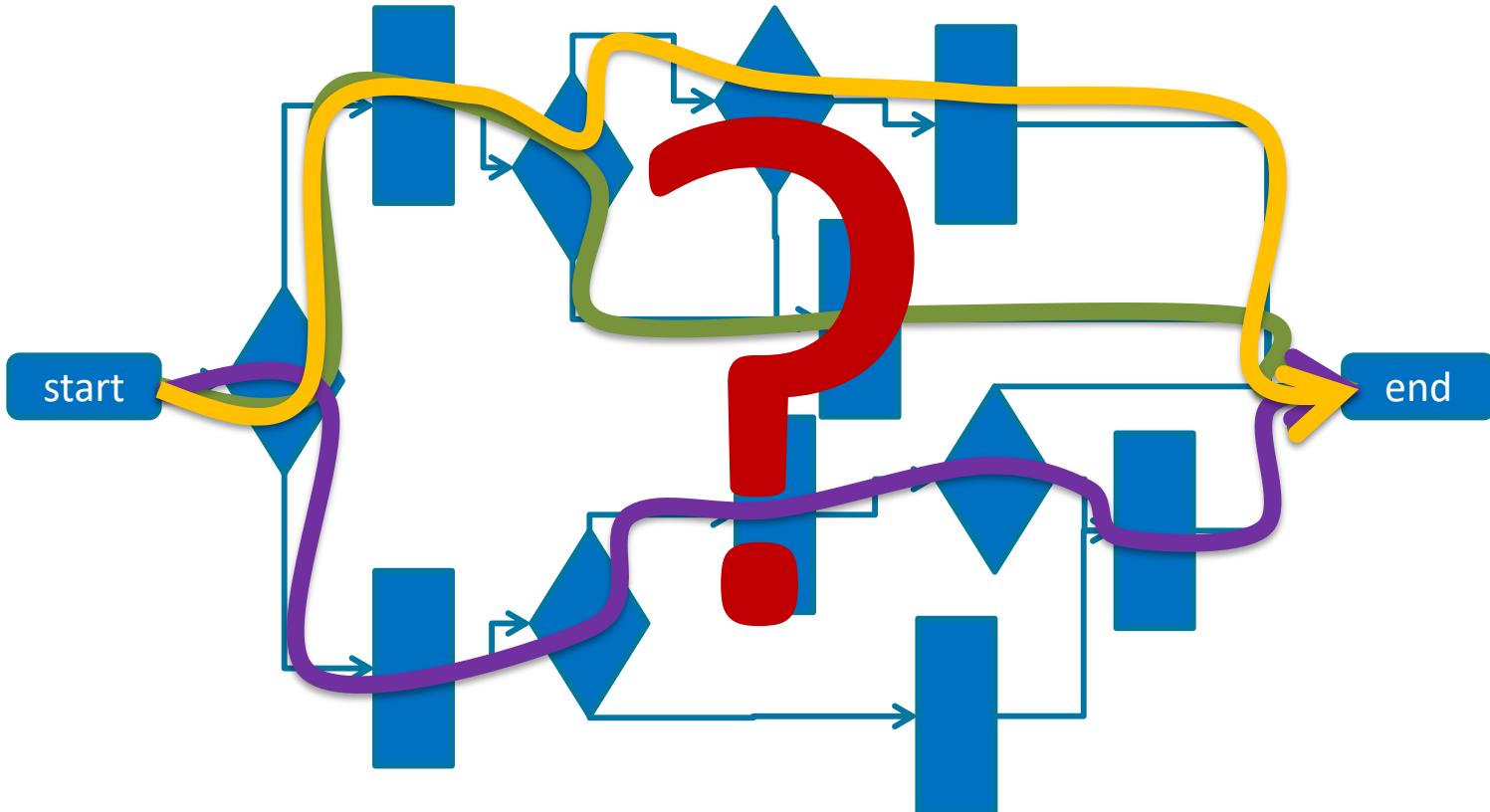
Sometimes using **break;** and
sometimes not is possible

```
public class Main{
    public static void main(String[] args){
        int position=1;

        switch(position) {
            case1:
                System.out.println("Your team won the tournament");
            case2:
            case3:
                System.out.println("Your team will move into the next division");
                break;
            case4:
            case5:
            case6:
                System.out.println("Your team will play in this division next season");
                break;
            case7:
            case8:
            case9:
                System.out.println("Your team will descend for the next season");
                break;
            default:
                System.out.println("Invalid position");
                break;
        }
    }
}
```



Tracing



Tracing (2)

Following the execution flow by hand

- Write out values of variables
- Evaluate conditions by hand
- Follow the control flow as defined by your program

Very informative procedure

Can be a lot of work, and next to impossible for large programs

Logging

Using **print** and **println** statements to track program executing

```
...  
if (choice == 5) {  
    System.out.println("Choice is 5");  
    ...  
}  
...
```

- need to be removed at the end
- there are ways around this

```
final boolean DEBUG = true;  
...  
if (choice == 5) {  
    if (DEBUG) System.out.println("Choice is 5");  
    ...  
}  
...
```



More on Scanner

Entering a line of input:

1 2 3 4 5 6 7 8 99 <enter>
↑

- `nextInt()`, `nextDouble()`, `next()` only read until space
- `hasNextInt()`, `hasNextDouble()`, `hasNext()` check to see if there is a next item (of the correct form) available
 - always returns a boolean: `true` or `false`

More on String

More methods can be called on Strings similar to `equals(String a)` and `compareTo(String a)`

- `startsWith(String a)` returns boolean
- `endsWith(String a)` returns boolean
- `length()` returns int
- `charAt(int index)` returns char; starts counting from 0 to `length()-1`!
- `substring(int begin, int end)` returns String; again counting from 0!

You can practice some of them in codingbat (warmup2)



Example

```
public class StringFun {  
  
    public static void main(String[] args) {  
  
        String a = "Apple";  
        String b = "Microsoft";  
  
        if (a.startsWith("App"))  
            System.out.println("Apple started with Apps.");  
  
        String c = b.substring(2,6);  
  
        System.out.print("The string " + c);  
        System.out.println(" contains " + c.length() + " characters");  
        System.out.print("of which " + c.charAt(c.length()-1));  
        System.out.println(" is the last.");  
    }  
}
```

```
wopr: java$ javac StringFun.java  
wopr: java$ java StringFun  
Apple started with Apps.  
The string cros contains 4 characters  
of which s is the last.  
wopr: java$ █
```

Problem with Scanner and Doubles

```
import java.util.Scanner; // Import the Scanner class
import java.util.Locale;

public class Main {
    static int a; // Global variable: Class level scope

    public static void main(String[] args) {
        double a; // Local variable: Method (main) level scope
        Scanner scanner = new Scanner(System.in);
        Locale locale = Locale.US;
        scanner.useLocale(locale);
        System.out.println("Enter a number: ");

        a = scanner.nextDouble();
        System.out.println("Your number was: " + a);

    }
}
```



Frequent Errors

1. Dangling else

```
if (a>500)
    if (b>500)
        System.out.println("a and b are big");
else
    System.out.println("Now which one isn't big?");
```

Correct syntax, but bad form!

Use { and } for clarity!

Frequent Errors (2)

2. < or <= ?

Don't forget to think when programming.

Don't simply use trial and error.

```
if (a < 2)    ≡    if (! (a >= 2))
```

Frequent Errors (3)

3. Incorrect use of && and ||

```
if (a == 2 && 3 && 5 && 7)
    System.out.println("a is prime and a < 10");
```

```
Main.java:6: error: bad operand types for binary operator '||'
      if(a == 2 && 3 && 5 && 7)
              ^
      first type: boolean
      second type: int
1 error
compiler exit status 1
```

```
if (a == 2 || a == 3 || a == 5 || a == 7)
    System.out.println("a is prime and a < 10");
```



Frequent Errors (4)

4. Confusing **&&** and **||** in long expressions

E.g. De Morgan's Law

$$\begin{aligned} \text{not}(A \vee B) &= \text{not}(A) \wedge \text{not}(B) \\ \text{not}(A \wedge B) &= \text{not}(A) \vee \text{not}(B) \end{aligned}$$

in Java

<code>!(a b)</code>	<code>(!a && !b)</code>
<code>!(a && b)</code>	<code>(!a !b)</code>

*Don't
forget
to
think!*

Short exercise

$$!(a \mid\mid b) == (\neg a \And \neg b)$$

$$!(a \And b) == (\neg a \mid\mid \neg b)$$

	$!(a \mid\mid b)$	$(\neg a \And \neg b)$	$!(a \And b)$	$(\neg a \mid\mid \neg b)$
a = true b = true	false	false	false	false
a = true b = false	false	false	true	true
a = false b = true	false	false	true	true
a = false b = false	true	true	true	true



Frequent Errors (5)

5. In decision lists, it is the combination of order and conditions that defines the semantics

```
if (numberOfBeers < 1)
    System.out.println("Feel free to drive");
else if (numberOfBeers < 8)
    System.out.println("Feel free to ride with me");
else if (numberOfBeers < 16)
    System.out.println("Try not to be sick on the bus");
else if (numberOfBeers < 32)
    System.out.println("I think you might have a problem");
else
    System.out.println("The paramedics have been called");
```

```
if (numberOfBeers < 32)
    System.out.println("I think you might have a problem");
else if (numberOfBeers < 16)
    System.out.println("Try not to be sick on the bus");
else if (numberOfBeers < 8)
    System.out.println("Feel free to ride with me");
else if (numberOfBeers < 1)
    System.out.println("Feel free to drive");
...
```

