Installation of Java

Starting Java with JShell

Instructions of how to install Java 11 on Windows

- We will install Oracle Java 11 on a Windows 10 machine
- Installation is only slightly different on other operating systems

- Within the course we use Oracle Java 9 for the demos
- You can use any Oracle Java beginning from Java 9 for practising.

Using JShell for simple Math
or
Operators and Variables

Java from Scratch

Operators

- \+ : Adds two values
- \- : Subtracts two values
- * : Multiplies two values
- / : Divide to values
  - for integers without remainder, for floating numbers it produces a floating number
- % : Gives the remainder for integers
- ( ) : Braces can be used to overwrite math rules

Variables in Java

- Think of a variable as a **container** which can hold a value, text or even an object.
- Each variable gets a **name**.
- Within the rest of the code the variables are used as **placeholder** for the real value, text or object.
- Java is a **strongly typed language**. That means, every variable needs a **type** before anything is stored in it.

Types for Variables for Whole Numbers

- int : negative and positive whole numbers like 1, -1 , 1000
  - 32 Bit from –2,147,483,648 to 2,147,483,647
- byte : the same, smallest range
  - 8 Bit from -128 to +127
- short : the same, still a small range
  - 16 Bit from -32,768 to 32,767
- long : if you need more than int
  - 64 Bit from –9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Types for Variables for Floating Numbers

- double : floating double numbers like 3.1415
  - 64 Bit for double precision from 4.9E-324 to 1.8E+308
- float : floating numbers with simple precision
  - 32 Bit for numbers from 1.4E-45 to 3.4E+38

How to assign a value to a variable in Java?

- Remember: Java is a strongly typed language. So the type of the variable must be set before the variable is set:
  - int number;
- After the variable is introduced this way, you can use the equal sign to assign a value to it:
  - number = 42;
- And you can do both things in one line:
  - int number = 42;

Strings, Chars and Booleans and their Operators

Java from Scratch

Variable Types: String, Char & Boolean

- String : Text values like "Hello World!"
- char : For characters like 'J' or 74, which is the ASCII-value for J
- boolean : Logical values true or false

String Concatenation

- Strings are **sequences of characters**
- They are written in **double quotation marks** like this:
  - "Hello World“
- The values can be **concatenated** with the **plus** sign:
  - String h = "Hello " + "World“;
- Note that the String is the only **non-primitive** variable type, we want to talk about here. It is actually an object. That means you have to start with a capital ‘S’

Equality checks result in Booleans

- You can assign **true** or **false** to a boolean character
  - boolean b = true;
- But you can also run an equation like this:
  - boolean b = i == 1;
- It checks if ‘i’ equals 1.
  - If it does ‘i == 1’ returns true and so b is true.

Equality Operators & Relational Operators

- **==** : **equal to** – equality operator
- **!=** : **not equal to** – equality operator
- **>** : **greater than** – relational operator
- **>=** : **greater than or equal to** – relational operator
- **<** : **lesser than** – relational operator
- **<=** : **lesser than or equal to** – relational operator

Conditional Operators

- Equality & relational operators can be combined with **conditional operators:**
  - **&&** : means **AND**
  - **||** : means **OR**

- E.g. **(i<10) && (j >0)** can be assigned to a boolean because it results in a boolean value.

Compound-Assignment Operators & Unary Operators

Java from Scratch

Compound-Assignment Operators (1)

- += assigns the result of the addition
  - **a += b** is the same as **a = a + b**
- -= assigns the result of the subtraction
  - **a -= b** is the same as **a = a - b**
- *= assigns the result of the multiplication
  - **a *= b** is the same as **a = a * b**
- /= assigns the result of the division
  - **a /= b** is the same as **a = a / b**

Compound-Assignment Operators (2)

- %= assigns the remainder of the division
  - **a %= b** is the same as **a = a % b**

- &= assigns the result of bitwise AND
  - **a &= b** is the same as **a = a & b** which is the same as **a = a && b** for us
- |= assigns the result of bitwise OR
  - **a |= b** is the same as **a = a | b** which is the same as **a = a || b** for us

Unary Operators

- i++ : Postincrement
- ++i : Preincrement
- i-- : Postdecrement
- --i : Predecrement

And one more boolean operator:

- ! : Negation NOT

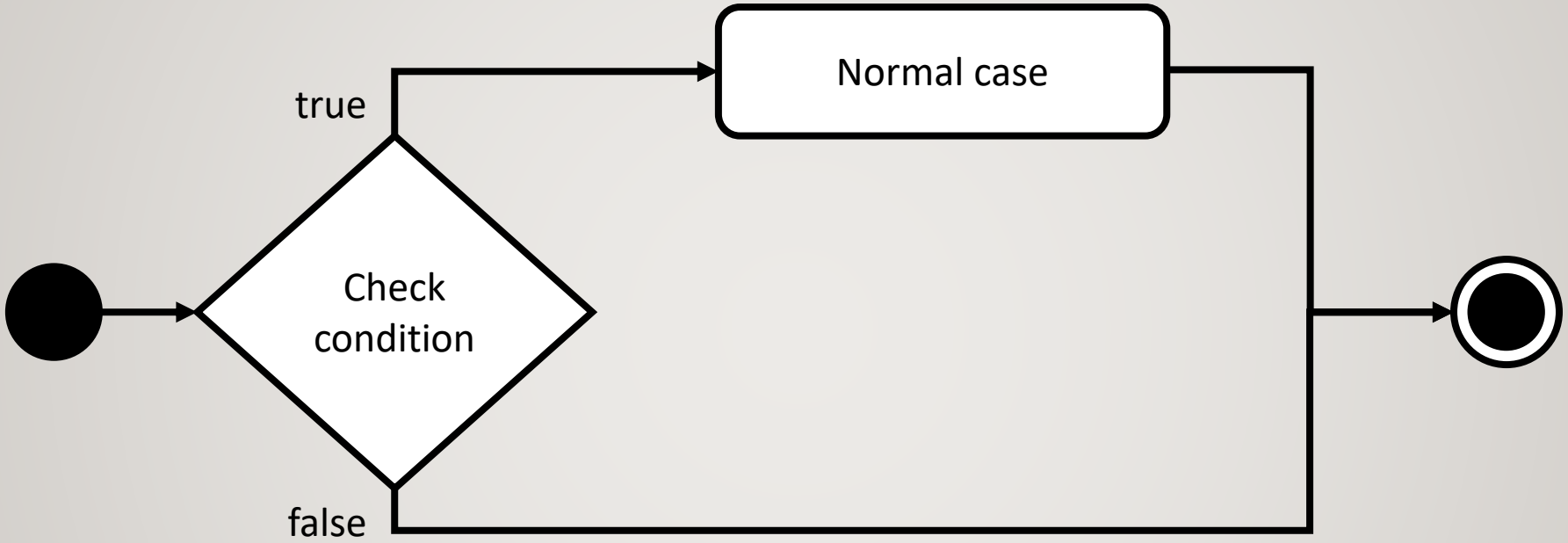
Summary

- Mathematical **operators**
- Types of **variables**
- Assignment of **variables**
- Strings, Chars, Booleans **variables**
- **Operators** for them
- Compound-Assignment **Operators**
- Unary **Operators**

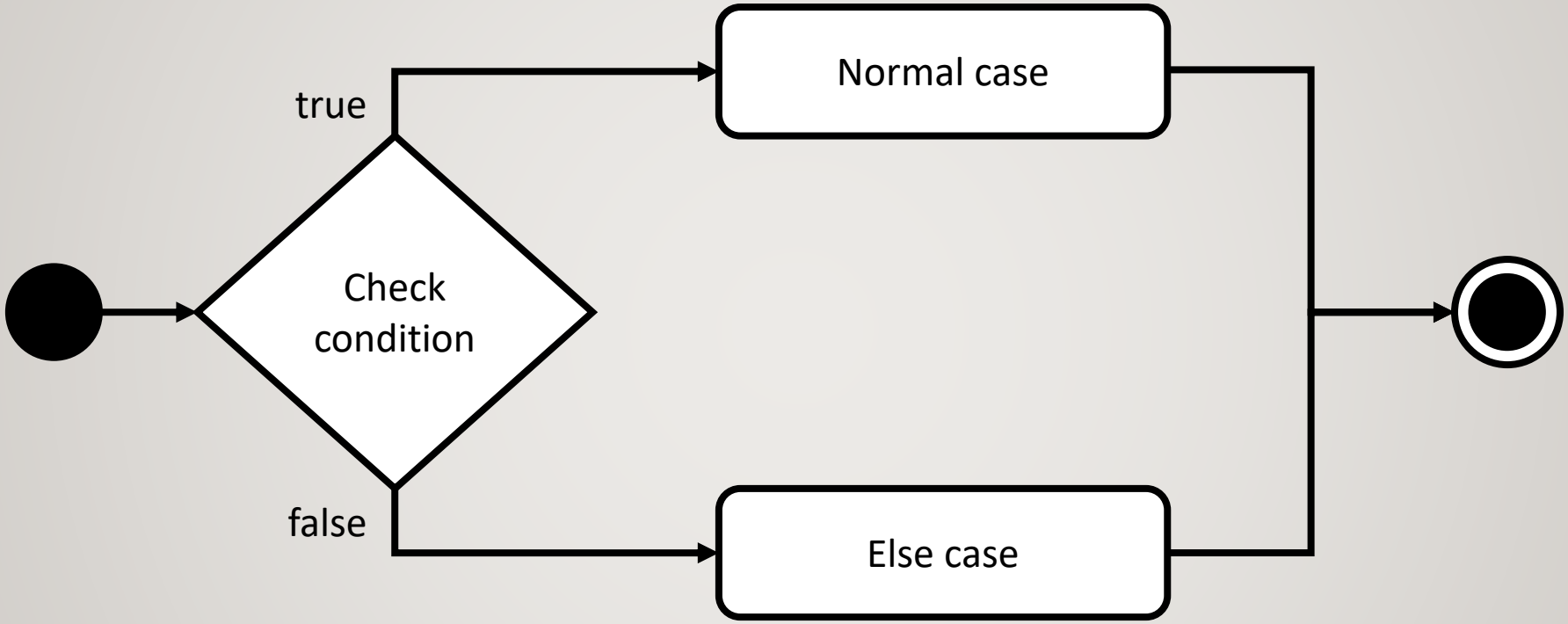
Running code depending on a condition: The If-Statement

Java from Scratch

If-Statement Visualization as Flow Chart



If-Statement Visualization as Flow Chart



The If-Statement in Java

```
if (i == 1) {
        System.out.println ("i is one");
    }
    else  {
        System.out.println ("i is something else");
    }
```

The If-Statement in Java


```
if (i == 1)
        System.out.println ("i is one");

    else
        System.out.println ("i is something else");
```

The If-Statement in Java


```
if (i == 1) {
        System.out.println ("i is one");
    }
    else
        System.out.println ("i is something else");
```

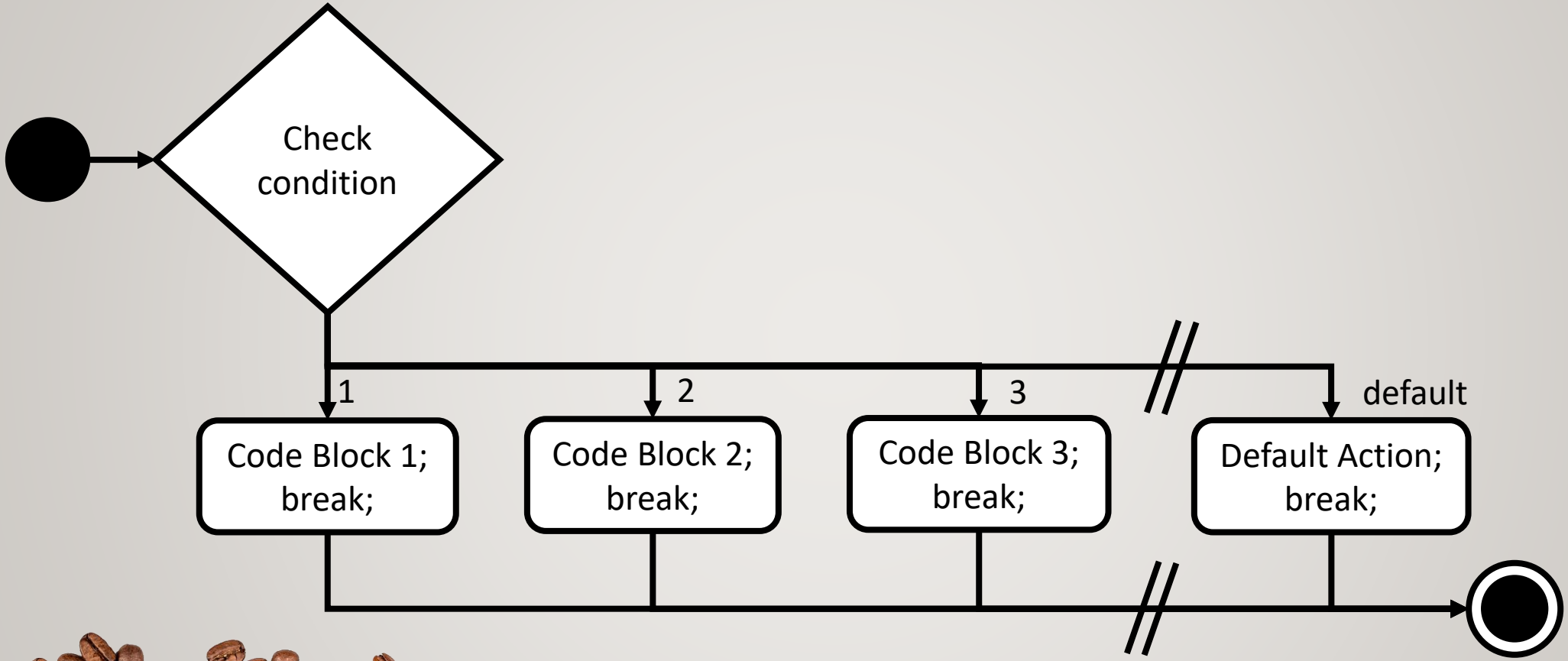
The Ternary Conditional Operator ? :

- The **Ternary Operator ? :** was introduced with Java 7 and is written this way
  - variable = condition **?** true-part **:** false-part;
  - If the **condition** is true, the **true-part** is evaluated and assigned to the variable, otherwise it is the **false-part**.
- So this gives the minimum of the two numbers a and b:
  - int minimum = (a < b) ? a : b;

Multiple Options: The Switch-Statement

Java from Scratch

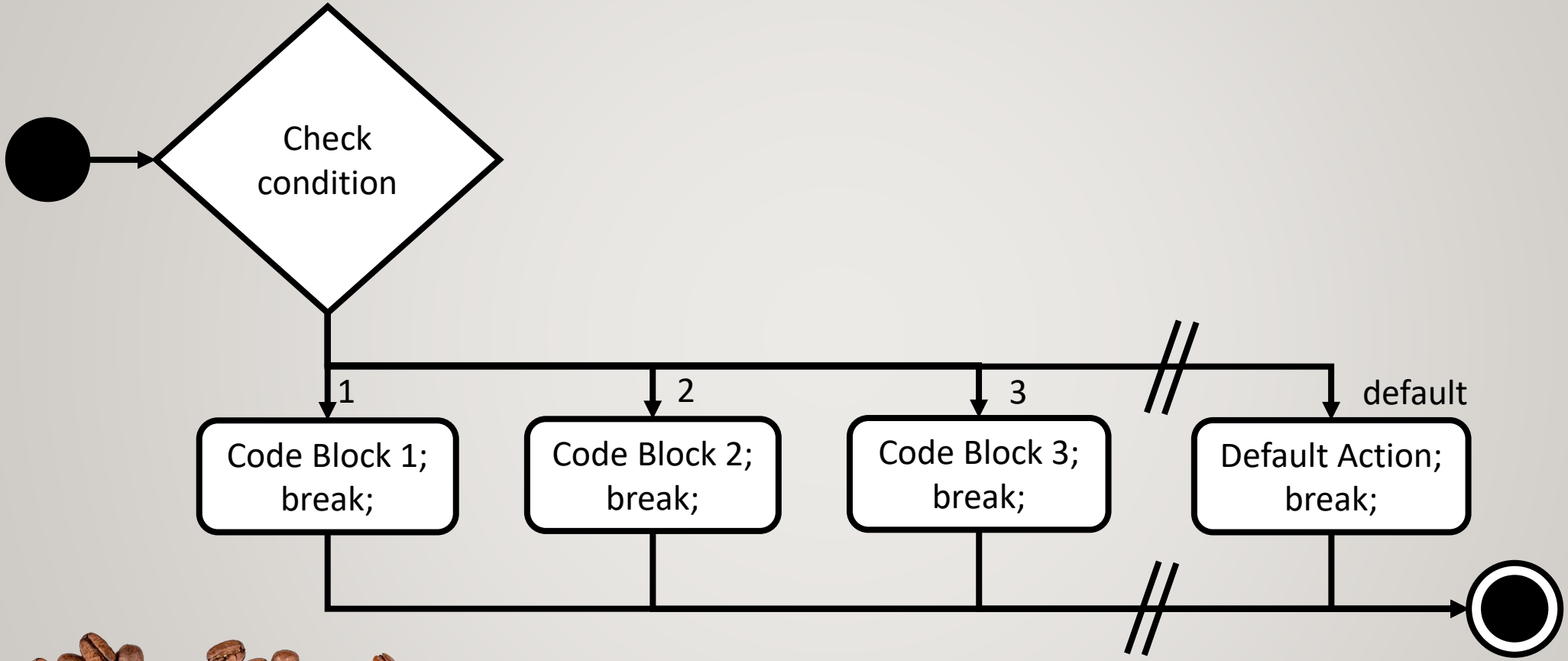
Switch-Statement visualized as Flow Chart



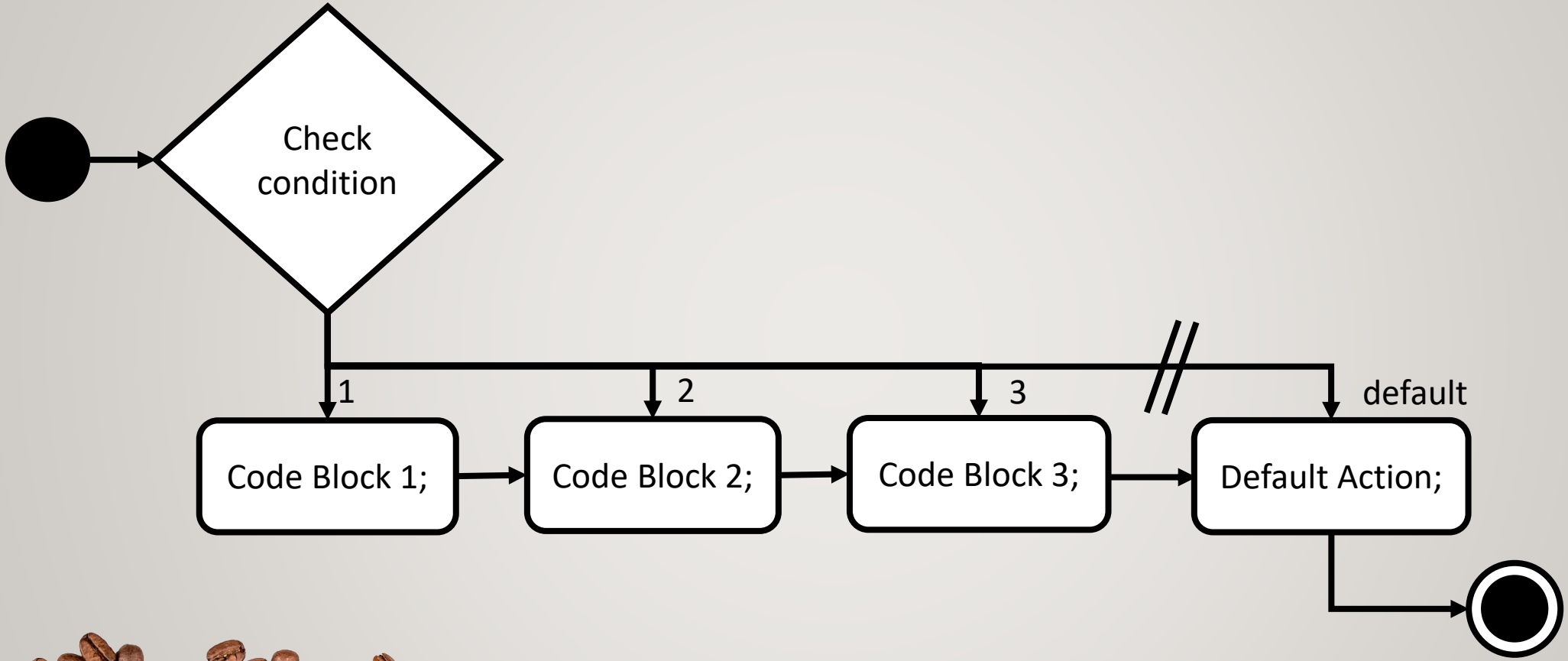
A exemplary Switch-Statement

```
int variable = 2;
switch (variable) {
    case 1:  System.out.println (“one”);
             break;
    case 2: System.out.println (“two”);
             break;
    default: System.out.println (“unknown”);
             break;
}
```

Never forget the break!



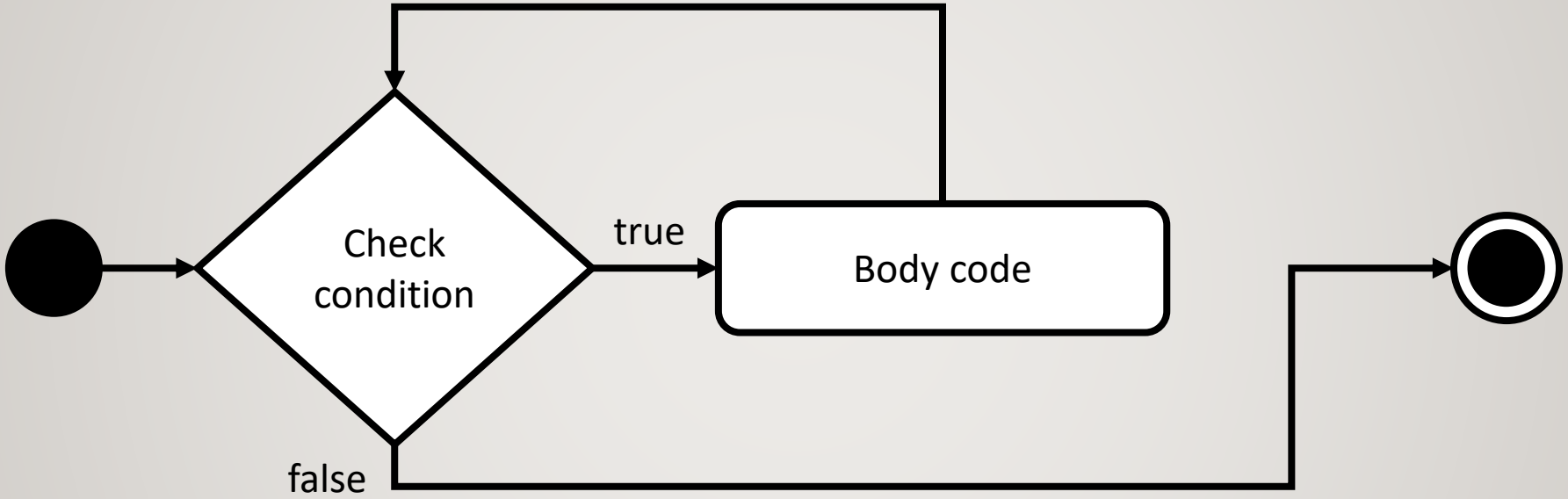
Never forget the break! This will happen!



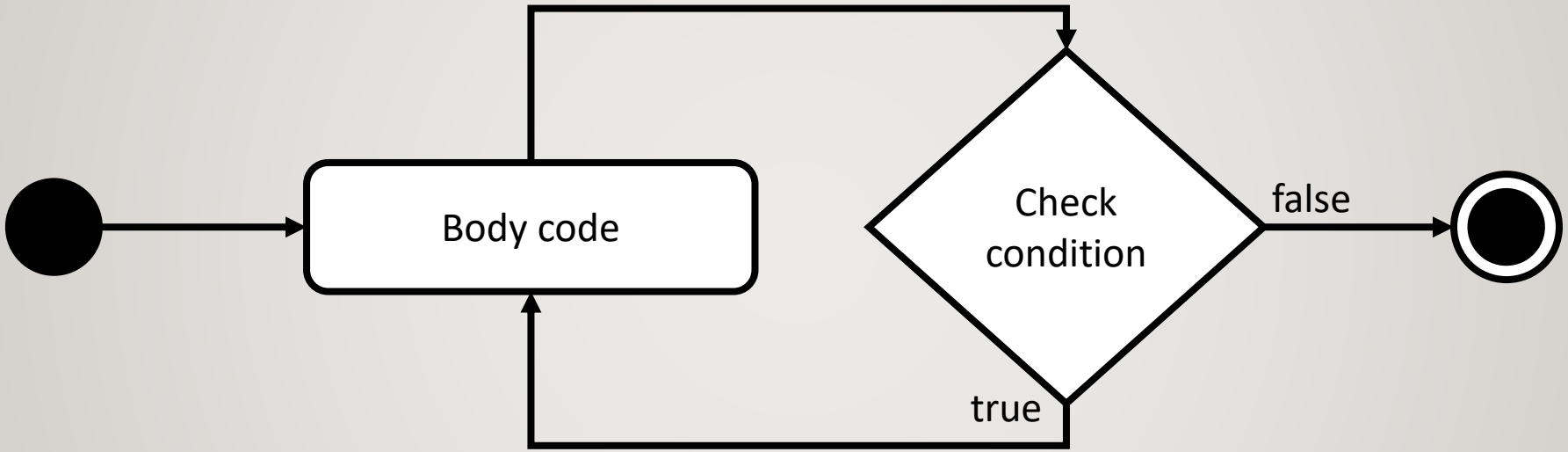
Running code repeatedly: The While-Loop and the Do-While-Loop

Java from Scratch

While-Loop visualized as Flow Chart



Do-While-Loop visualized as Flow Chart



While-Loop with Curly Braces and without

- ```
  int i = 0;
  while (i < 10) {
          i++;
  }
  ```

- ```
  int i = 0;
  while (i < 10)
          i++;
  ```

Do-While-Loop with Curly Braces and without

- ```
  int i = 0;
  do {
          i++;
  }
  while (i < 10);
  ```

- ```
  int i = 0;
  do i++;  while (i<10);
  ```

Multiple Values of the same Type: Arrays

Java from Scratch

Arrays are Collections of Variables

- Often you need a **collection of variables** of the **same type**
  - This variables can be put into an array
- An array is simply a **continuous list in memory** which consists of all variables **one by one**.

- The length of the array must be known **in advanced**, because Java needs to reserve memory for the array.
- You **cannot change the length** of an array.
- In Java the length of an array is limited to the **maximum int-number**. But that is a lot of space. To be precise: 2,147,483,647

Two Ways to define an Array

- **Preinitialized** it the moment the variable is defined:
  - int [] integerArray = { 1, 2, 3, 4, 5 };
  - String [] stringArray = {"blue", "green", "red"}
- Or **assign** it after the variable was defined:
  - int [] integerArray;
    integerArray = new int [5];
  - String [] stringArray;
    stringArray = new String [3];

Changing and Accessing Values

- Arrays begin at position 0. If you define an array of length 8, this is the resulting structure:

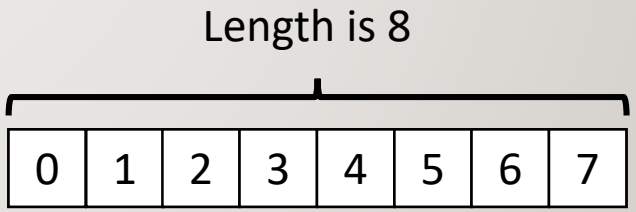
0	1	2	3	4	5	6	7

- You **put a value** into the array by determine its position:
  - integerArray [3] = 7;
    stringArray[2] = "white";
- You **get a value** out of the array by accessing its position:
  - integerArray [3]
  - stringArray[2]

Beyond the Limits

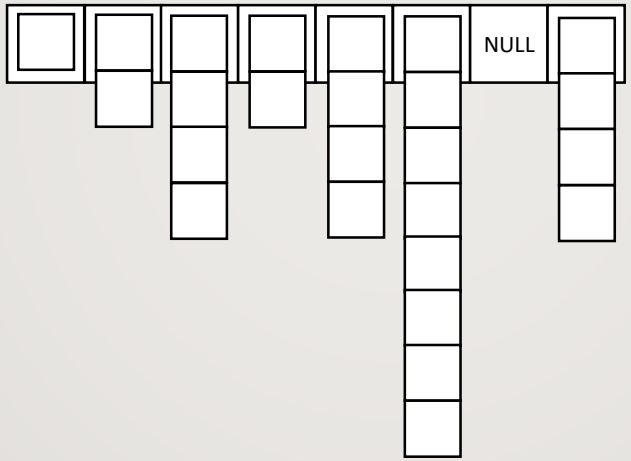
- But if you want to put a value into a position which is not available…
- … or you want to read a position which is not available…
- …you will get an **ArrayOutOfBoundsException** at runtime.

- To determine the length of an array you use length:
  - stringArray.length
  - integerArray.length



Multidimensional Arrays

- Multidimensional Arrays are implemented as **Arrays in Arrays**
- A two-dimensional Array can have a look like this:



Multidimensional Arrays

- **Definition** of a two-dimensional array with two squared braces:
  - int [][] integerArray = { {1,2}, {1,2,3}, {1,2,3,4}, {1,2,3,4,5}, {1,2,3,4,5,6} };
- **Access** of a two-dimensional array with two squared braces:
  - integerArray[0][1]

- If you want to have more dimensions you have to add more squared braces.

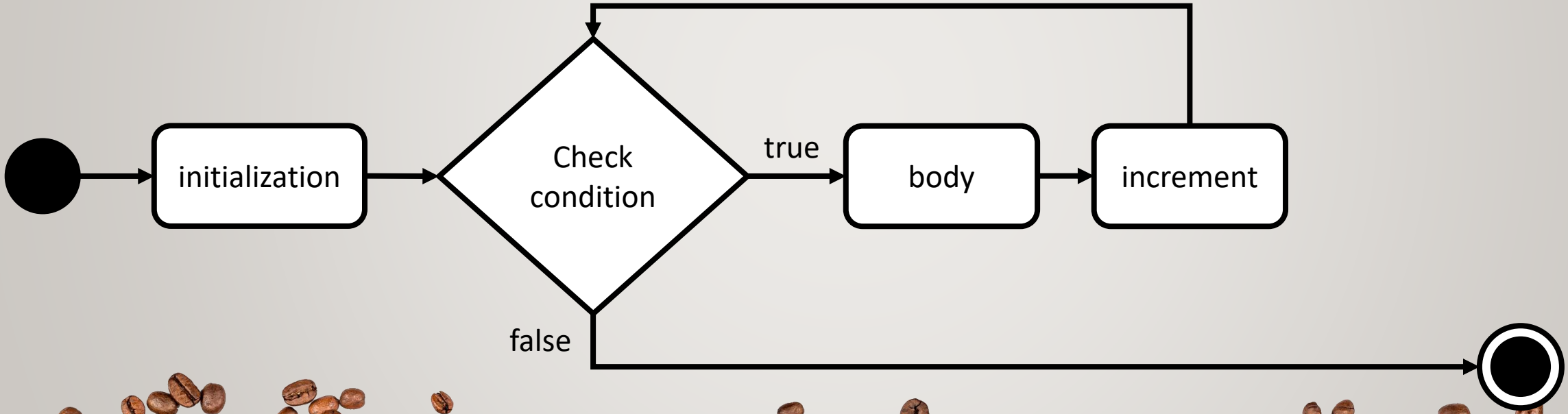
Running code a specific number of times: The For-Loop

Java from Scratch

For-Loop visualized as Flow Chart

```
for (initialization; check; increment) {
   body;
}
```

```
for (int i = 0; i < 100; i++) {
   System.out.println (array[i]);
}
```



The For-Loop

- for (int i = 0; i < array.length; i++) {
  // do something with array[i]
  }

- for (Iterator iterator = collection.iterator(); iterator.hasNext();) {
  Item item = (Item) iterator.next();
  // do something with item
  }

The For-Each-Loop

- for (Item item : collection) {
  // do something with item…
  }

- Came with Java 5
- collection must return an iterator

Curly Braces

It is possible to write any for-loop without Curly braces:

- for (int i = 0; i < array.length; i++) {
  command(array[i]);
  }

… is the same as …

- for (int i = 0; i < array.length; i++)
  command(array[i]);

Break & Continue

- With break you can jump out of a statement completly.
  - We saw this for the Switch-statement. But is is also possible with While- or For-loops.
- With continue you can skip the current run

- For both it is possible to specify a destination target
  - continue target;

    … jumps to …
  - target:

Methods: The Java-Way to create Subprograms

Java from Scratch

Methods in Java

- Methods are **subprograms**
  - In other programming languages methods are named **functions**, **procedures** or **subroutines**.
- Code is taken out from the main program and can **invoked at arbitrary positions** in the program.
- It can be invoked once, **multiple times** or never.
- **Arguments** can be passed into the method and the method may have a **return parameter**.

Declaration of a Method (1)

- Methods can be **declared** this way:

```
returntype methodname (paramtertype parametername, …) {
    // use parametername
    return variableofreturntype;
}
```

- The list of **passing parameters** is arbitrary long:
  - int methodWithOneParameter (int parameter1) {…
  - int methodWithTwoParameters(int parameter1, String parameter2) {…
  - int methodWithNoParameter() {…

Declaration of a Method (2)

- There can be **one** return value or **no** return value
  - int methodWithOneReturnValue () {
    int i = 1;
    return i;
    }
  - void methodWithnoReturnValue () {
    return;
    }
- If there's no return value writing **return;** is optional.

An Example of a Method

- An example method:

```
int sumOfTwoInts (int a, int b) {
    int r = a + b;
    return r;
}
```

- Within the main program we can **invoke** the method by writing e.g.:
  - int sum = sumOfTwoInts (1,2);

Methods with variable Parameter Lists

- There is a language feature to allow **variable parameter lists**.
- This can be used with the **last** parameter:
  - void foo(String a, int... b) {
    if (b.length > 0) System.out.println (b[0]);
    if (b.length > 1) System.out.println (b[1]);
    System.out.println (a);
    }
- Note that within this program either 0, 1 or 2 int-parameters can be passed. More parameters can be passed but are ignored.

Summary

- If-Statement
- Switch-Statement
- While-Loop, Do-While-Loop
- For-Loop, For-Each-Loop
- break; and continue;
- Methods

Practical Exercise

- Develop a compound interest calculator - method
- Header of the method should be:
  - double calculateResult (double startAmount, double interestRate, int years)
- The percentage of the interest rate should be given divided by 100 e.g. 3% is 0.03
- The final balance should be returned.

Some more cool features of JShell

- You can write out the complete history to a file using the save-command:
  - /save filename
  - This is useful if you want to hand in your function
- You can also read this file and execute it
  - /open filename

Pause the video now
to do the exercise

If you didn‘t manage to create the method, here it is:

```
double calculateResult (double startAmount, double interestRate, int years) {
        for (int i = 0; i< years; i++) {
                startAmount *= (interestRate+1);
                System.out.println ((i+1)+ " "+startAmount);
        }
        return startAmount;
}
```

A Java-file gets executed

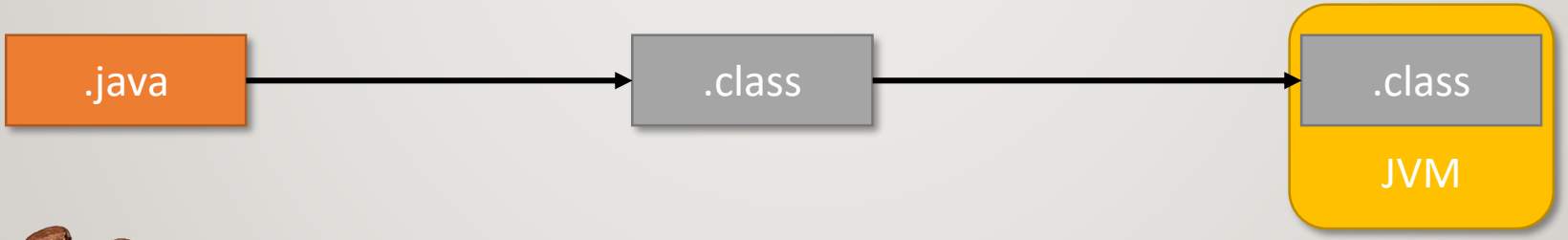
Java from Scratch

What is a Java-file?

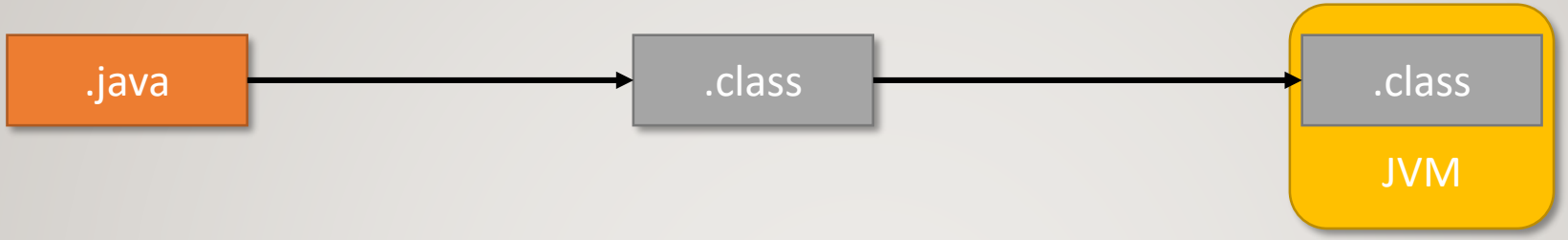
- Up to now we only worked with JShell, a REPL-console (**R**ead–**e**val–**p**rint **l**oop)
- We can write Java-programs as text files, compile and execute them.
- Let us write a small Java program, which prints out **“Hello world!”**

What steps are needed for a Java program to run?

- Every Java-program is actually build out of a set of **text-files** with the ending **.java**.
- These .java files are compiled to **bytecode-files** by a Java-compiler to **.class-files**.
- After this the .class-files can be run on a **J**ava **V**irtual **M**achine (JVM).

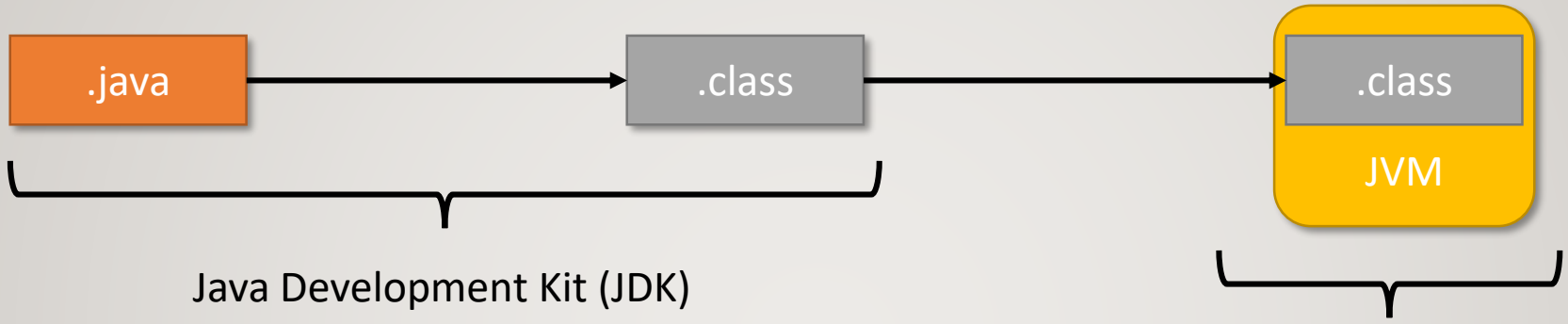


Why do we need a Java Virtual Machine?



- You may ask yourself: Why don‘t we directly have machine code which is directly executable?
- Java bytecode is translated on the fly to machine code while running the program but gives us the flexibility to run this code on different CPU architectures.

What is a JDK (a Java Developer Kit)?



- You need a Java Developer Kit (JDK) to **build / compile** Java programs.
- You need a Java Runtime Environment (JRE) to **run** Java programs. (included with the Java Development Kit (JDK))

Download and Installation of the Java Development Kit (JDK)

- Both are available for Windows, Linux, Mac and other platforms.
- Both are available from multiple vendors.
- We choose the vendor **Oracle** and download & install the Java Development Kit (JDK) for **Windows**

Installing the IDE –
the Integrated Development Environment

Java from Scratch

Our current state

- Every Java program is actually **only a set of text files**.
- You can edit them with **every text editor** and start the execution of them from the command line.
- But this is quite uncomfortable.
- An **Integrated Development Environment (IDE)** makes the life of a software developer much more comfortable.

What is an **I**ntegrated **D**evelopment **E**nvironment?

- It is first a **text editor** and an invoker of the **compiler** and executor
- But it also helps to **navigate between the text files**
  - By understanding the content of the files, very good navigation is possible.
- By knowing the Java API and your code, it knows how to **complete typed statements**.
- It is even possible to **rewrite the program code automatically** (This is called Refactoring).

Different IDEs (1)

- **NetBeans**
  - Very easy to install. It is possible to download it together with the JDK.
  - Free
  - Has a plugin architecture. But not many plugins are provided.
- **IntelliJ IDEA**
  - Upcoming new star in the area of IDEs
  - Has a free community version. But you pay for more features.
  - Has a great set of plugins.

Different IDEs (2)

- **Eclipse**
  - Free & OpenSource
  - Great community: Many subprojects & Many plugins
  - Stable
  - In my eyes still the best IDE
- And there are **some other IDEs** on the market which are not big players.

Our plan now

- Prerequisites: Installed JDK 9 on your machine.

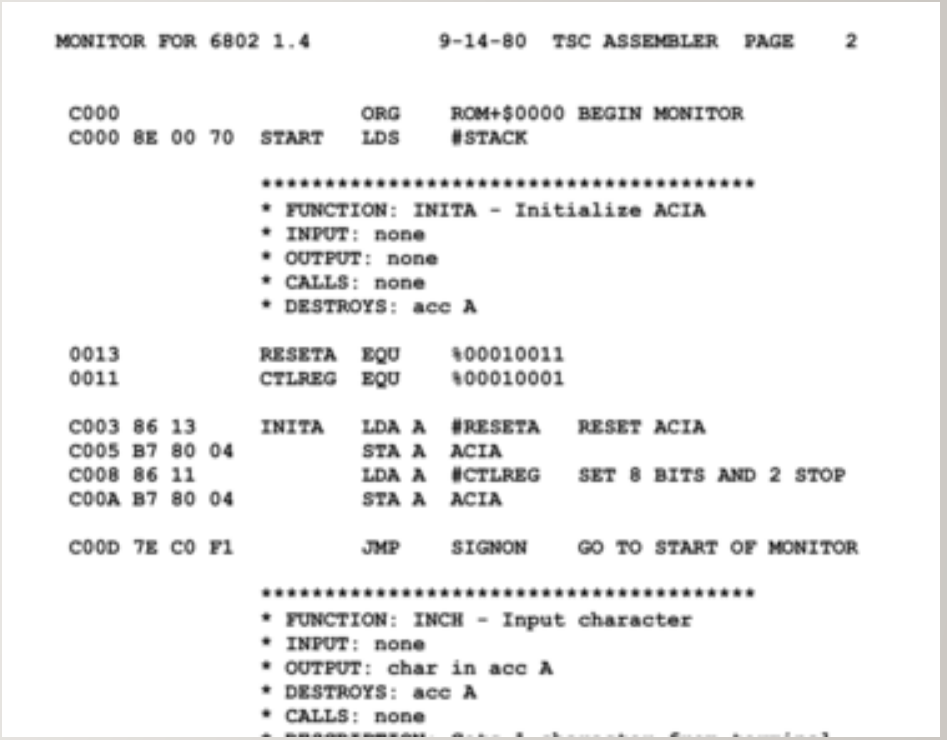
1. **Download** Eclipse
2. **Install** Eclipse
3. **Run** Eclipse & Create and Run the **HelloWorld-Program**.

Classes & Objects – Their variables and methods

Java from Scratch

Evolution of Programming Languages: The early Beginning

- Machine code:
  - Early programming happened in machine code.
  - Commands of the CPU where addressed by numbers.
- Assembler languages:
  - Later the “Assemblers” came up. They offer a more readable version of the programming with numbers.
  - Programmers need detailed knowledge of the hardware they program on.



```
MONITOR FOR 6802 1.4          9-14-80  TSC ASSEMBLER  PAGE    2


C000                          ORG    ROM+$0000 BEGIN MONITOR
C000 8E 00 70  START          LDS    #STACK

                    ****************************************
                    * FUNCTION: INITA - Initialize ACIA
                    * INPUT: none
                    * OUTPUT: none
                    * CALLS: none
                    * DESTROYS: acc A

0013                RESETA    EQU    %00010011
0011                CTLREG    EQU    %00010001

C003 86 13          INITA     LDA A  #RESETA   RESET ACIA
C005 B7 80 04               STA A  ACIA
C008 86 11                  LDA A  #CTLREG   SET 8 BITS AND 2 STOP
C00A B7 80 04               STA A  ACIA

C00D 7E C0 F1               JMP    SIGNON    GO TO START OF MONITOR

                    ****************************************
                    * FUNCTION: INCH - Input character
                    * INPUT: none
                    * OUTPUT: char in acc A
                    * DESTROYS: acc A
                    * CALLS: none
```

© WikiPedia

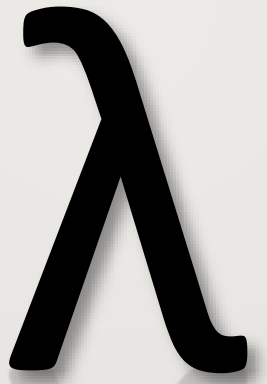
Evolution of Programming Languages: The Process-oriented Model

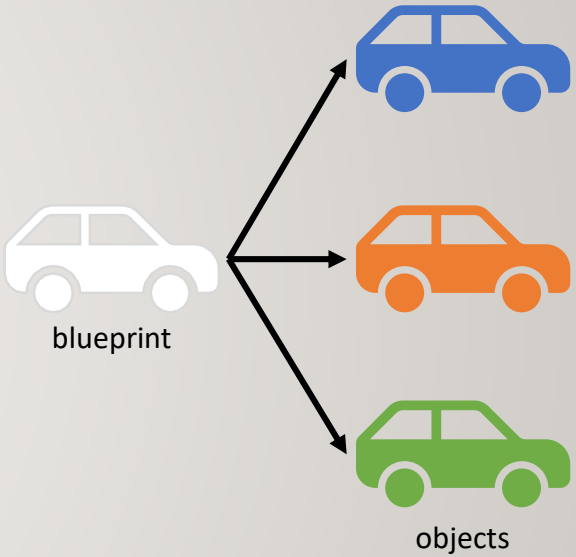
- Imperative languages
  - Give commands on a higher level.
  - There's only one main program which commands are executed one by one.
- Procedural languages
  - Subroutines are introduced.
  - A subroutine takes arguments and returns back a result.

© WikiPedia

Evolution of Programming Languages: The Object-oriented and functional model

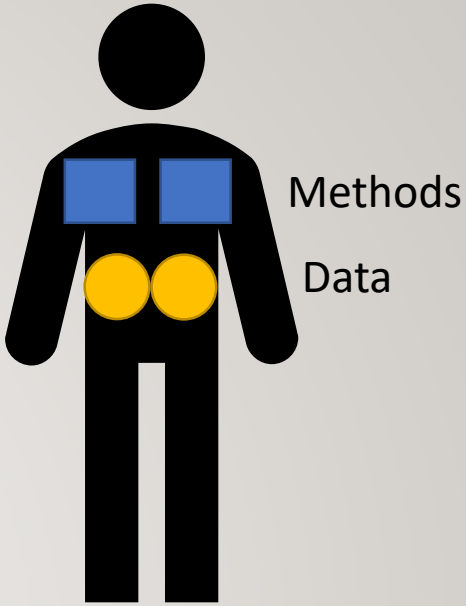
- Object-oriented languages
  - like Java
- Functional languages
  - Java 8 got a functional part

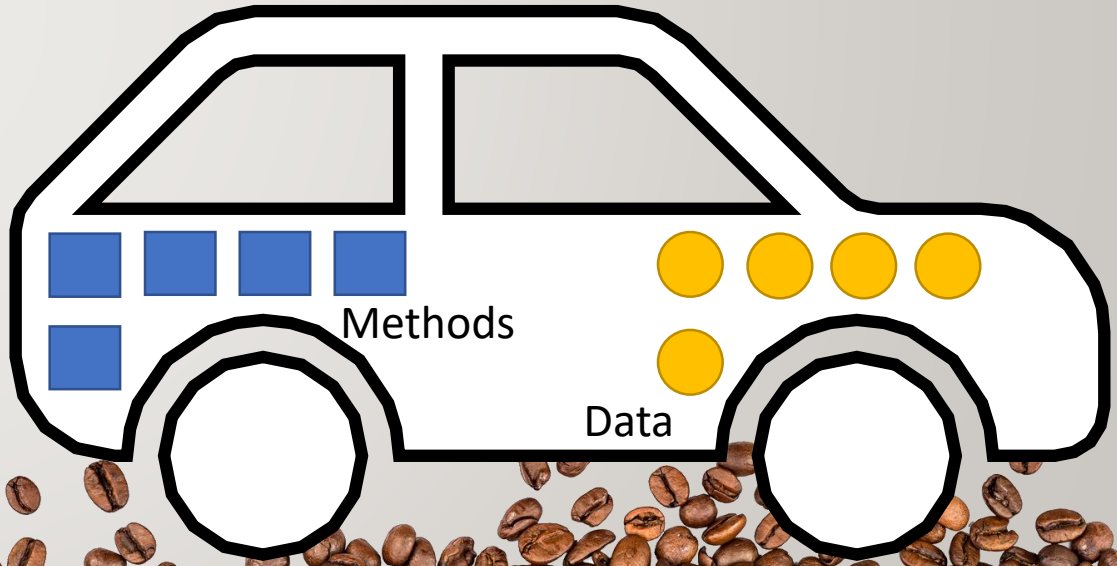




Variables and their Methods

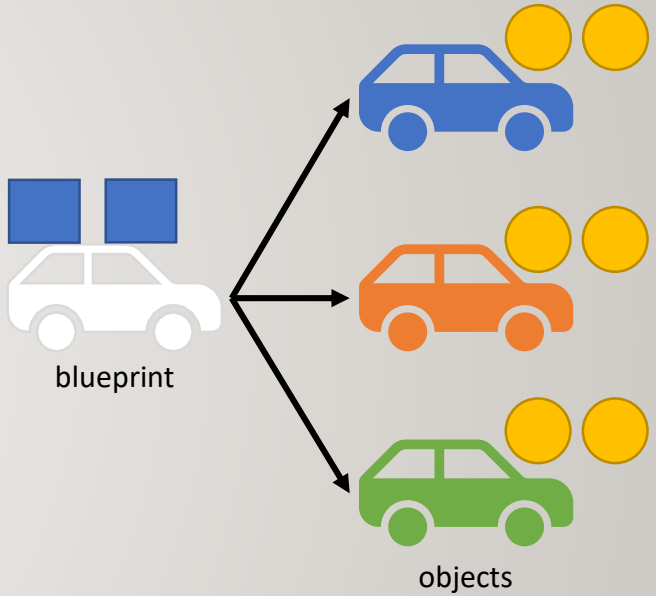
- In the world of computer programs we came to know that methods (procedures) and data **always go together**.
- The **same methods** and the **same type of data** is bounded to similar objects.
  - E.g. a person must have a first name and a last name. You can hire a person.
  - But a car doesn‘t have a first name and a last name. It has a serial number! And you can buy it. You do not hire it.
- Methods and data are **bounded together**. They are no separate things.
- This is the idea of **Object Oriented Programming (OOP)**





The Class is the Template of the Object

- It is possible that multiple objects of the same type are created.
  - Blue, orange and green car
- Each object has its own variable set…
- …but the methods over all the objects remain the same.
- An object is created out of a class:
  - The class is the template of the object.
  - The methods remain in the class.
  - The variables are created for the object.



A Class in Java

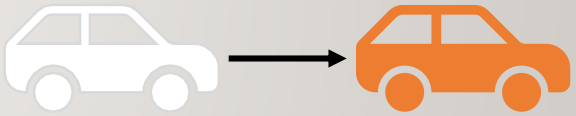
```
class Car {
        int serialNumber;
        String color;

        void drive (int kilometers) {
        }
        String getInformation (String parameter1) {
                return someInformationString;
        }
}
```

Car
serialNumber : int color : String
drive (kilometers : int) : void getInformation (parameter1 : String) : String

Instantiate an Object in Java

Creating an object with “new” and accessing the first class variable and invoking the first method:

```
Car myCar = new Car();
myCar.serialNumber = 1234567;
myCar.drive(100);
```



Learning Materials

Java from Scratch

Get help on the Internet

- Starting programming is not easy. It takes a lot of time to become a programmer.
- Learning is not happening automatically. You have to spend time and effort.
- So, do not panic. Do not give up. Ask someone.
  - Use the forum system inside the course.
  - There are many forums available on the Internet and beginners are welcome.

Internet pages

- Facebook group for help: https://www.facebook.com/groups/Javagroup123/ or https://www.facebook.com/groups/java.for.life/ or other
- Stack Overflow: https://stackoverflow.com/
- And others, maybe even in your mother tongue!
  - Use Google

How to ask good questions?

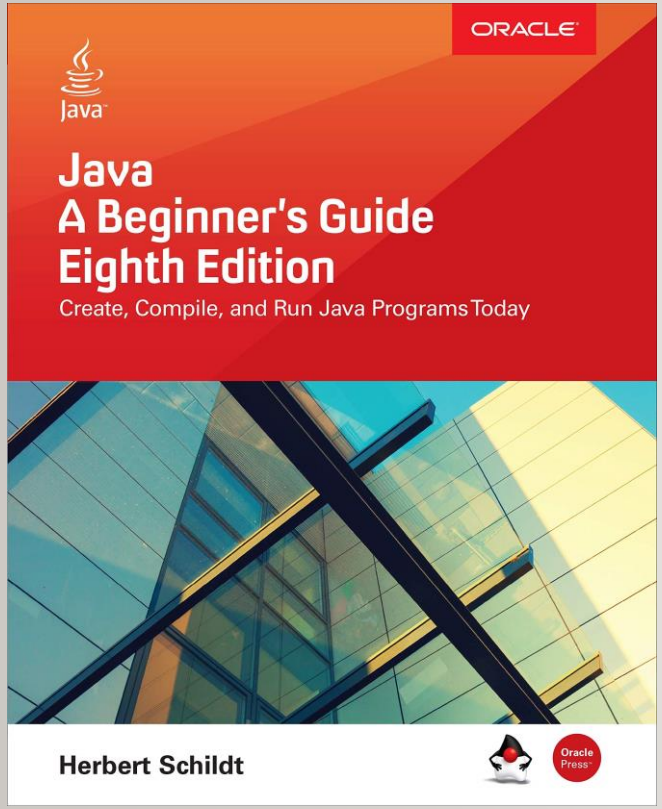
When you ask a question, do not simply drop a note: “It doesn’t work.”
Be precise:

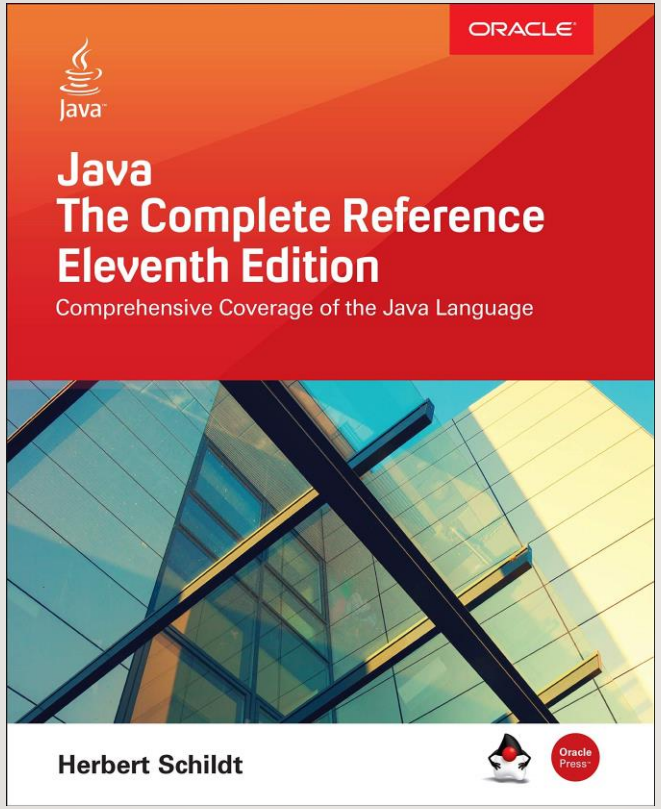
- What exactly doesn’t work? Post compiler errors or runtime stacktraces.
- Post the code which doesn’t work.
- Give information about the surroundings: Your operating system, Java version you use, IDE you use…

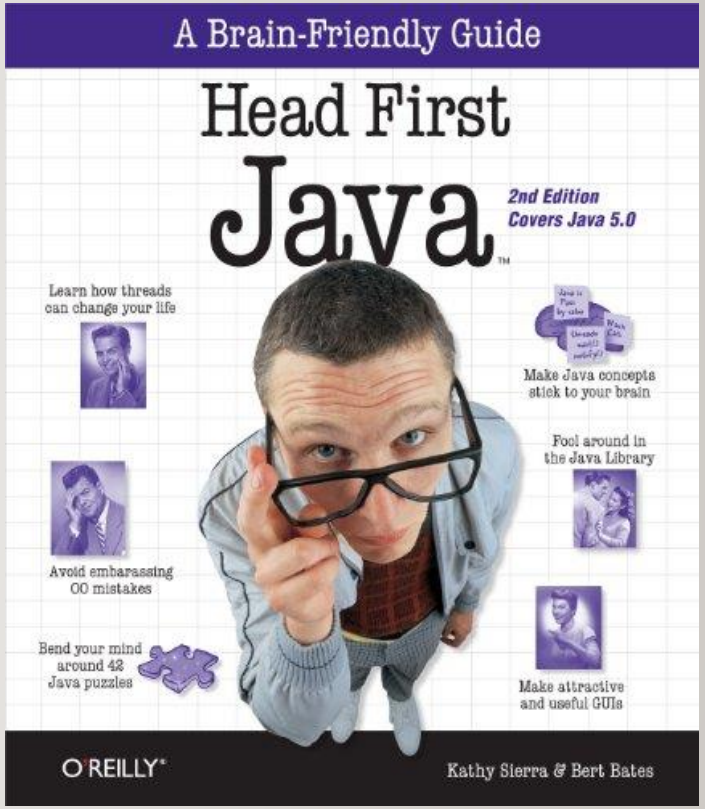
Internet pages with more information

- GoJava https://go.java
- The official Oracle Java Tutorials: http://docs.oracle.com/javase/tutorial/
- Use the API Docs http://download.java.net/java/jdk9/docs/api/overview-summary.html

Books


ORACLE
Java
Java
A Beginner's Guide
Eighth Edition
Create, Compile, and Run Java Programs Today
Herbert Schildt
Oracle Press


ORACLE
Java
Java
The Complete Reference
Eleventh Edition
Comprehensive Coverage of the Java Language
Herbert Schildt
Oracle Press


A Brain-Friendly Guide
Head First
Java
2nd Edition
Covers Java 5.0
Learn how threads can change your life
Make Java concepts stick to your brain
Fool around in the Java Library
Avoid embarassing OO mistakes
Bend your mind around 42 Java puzzles
Make attractive and useful GUIs
O'REILLY
Kathy Sierra & Bert Bates