Computer Science 1

Lecture 2

d.camporaperez@maastrichtuniversity.nl

Variables - Learning Goals

- You know the difference between a constant and a variable and know when to use which one
- You know how to name, declare, and instantiate variables
- You can use variables in your code effectively using operators
- You know properties about variables and constants such as data types and scope
- You know about casting, how and when to use it. Pitfalls and possible errors.

Variables and Values

- A **variable** is a place in memory that we can reference
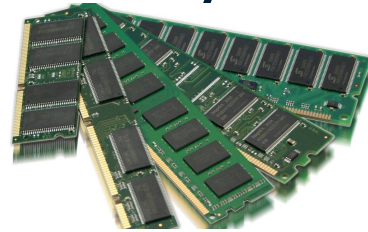
- A **value** is what a variable can hold

Data Type   Variable   Value

```
int x = 3;
```

Assignment operator

Maastricht University
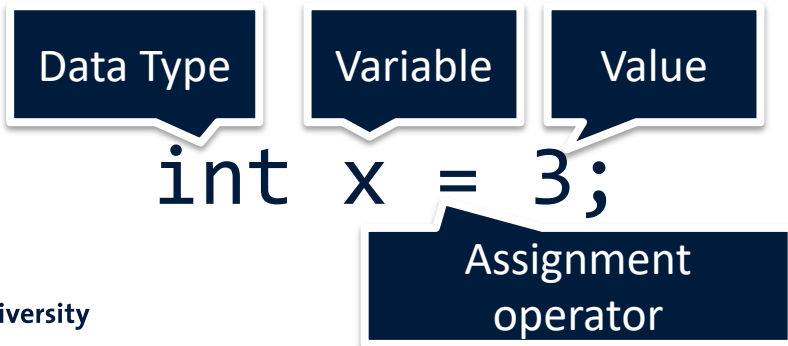
Variables - Operators

Operators are operation we can perform on variables and values.

1. Parentheses **()**
2. Unary operators **++ -- !**
3. Multiplicative operators *** / %**
4. Additive operators **+ -**
5. Relational ordering **< > <= >=**
6. Relational equality **==**
7. Logical and **&&**
8. Logical or **||**
9. Assignment **=**

Maastricht University

Variables - Declaration

To use a variable, you have to declare it.

Java naming convention[1]: **_camelCase_** or **_PascalCase_**

```
datatype variableName,…,variableName;
```

```
int i,j,k;
int numberOfStudents;
double temperature, volume, pressure;
String name;
```

1. https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html

Maastricht University

Variables - Expressions

Syntactically correct combinations of **variables**, **constants**, **operators**, **method invocations** and **values** that *evaluate* to a **single value.**

For instance:
```
b * 3 + c;
!a && (b < 3);
c = Math.pow(a, b) + 5;
```

Exercise together – What can we print? (2a)

- Create a file with a <class name> and .java extension.
- Create a public class <class name> inside the file, with a public static void main(String[] args) method.
- Use the operation System.out.println() to print:

  She said "Hello!" to me.
  12 + 20 is 32

Maastricht University

Variables - Casting

Converting the type of a value of an expression to a new type.

```
int a = (int)(3.14 * 5);        (a is now 15)
double b = 3.14;
int c = (int)(b * 12.3);        (b is now 38)
```

Maastricht University

Narrowing and widening data types

- We can distinguish between narrowing a datatype:

```
int a = (int)(3.14 * 5);
```

- And widening the datatype:

```
double a = (double)(3 * 5);
```

Beware of casting – Ariane 5, 1996

On 4 June 1996, the maiden flight of the Ariane 5 launcher ended in a failure. Only about 40 seconds after initiation of the flight sequence, at an altitude of about 3700 m, the launcher veered off its flight path, broke up and exploded.

The failure of the Ariane 501 was caused by the complete loss of guidance and attitude information 37 seconds after start of the main engine ignition sequence (30 seconds after lift-off). This loss of information was due to specification and design errors in the software of the inertial reference system.

*The internal SRI software exception was caused during execution of **a data conversion from 64-bit floating point to 16-bit signed integer value**. The floating point number which was converted had a value greater than what could be represented by a 16-bit signed integer.*

Exercise together – Casting exercise (2b)

- What are the results of...

```
11/3
11/3.0
11.0/3.0
(4 == 4.0) && (-5.6 < 0)
'a'+10
"a"+10
```

Maastricht University

Exercise together – Integer limits (2b)

- What are the integer limits?
- What happens if we try to represent a too big number in an int?
- And if we add two numbers surpassing that limit?

Constants

- Allows the use of a name for a memory location with a **fixed** value (i.e. can not be changed by the program). Use to avoid "Magic values"

- Use the `final` keyword
  e.g. `final double BOILING_POINT = 100.0;`

- Naming guidelines: **ALL_CAPITALS**

Why use constants?

```java
public class HelloWorld {

    public static void main(String[] args) {

        double width = 30, depth = 40;
        double area = width*depth;

        System.out.println("Price of lot = " + area * 500);

    }

}
```

Magic number

```java
double width = 30, depth = 40;
double area = width*depth;

System.out.println("Price of lot = " + area * PRICE_PER_SQUARED_METER);
```

Why use constants? (2)

```java
System.out.println("Price of lot 1 = " + area1 * 500);
System.out.println("Price of lot 2 = " + area2 * 500);
System.out.println("Price of lot 3 = " + area3 * 500);
System.out.println("Price of lot 4 = " + area4 * 500);
System.out.println("Price of lot 5 = " + area5 * 500);
System.out.println("Price of lot 6 = " + area6 * 500);
System.out.println("Price of lot 7 = " + area7 * 500);
```

```java
final double PRICE_PER_SQUARED_METER = 500;

System.out.println("Price of lot 1 = " + area1 * PRICE_PER_SQUARED_METER);
System.out.println("Price of lot 2 = " + area2 * PRICE_PER_SQUARED_METER);
System.out.println("Price of lot 3 = " + area3 * PRICE_PER_SQUARED_METER);
System.out.println("Price of lot 4 = " + area4 * PRICE_PER_SQUARED_METER);
System.out.println("Price of lot 5 = " + area5 * PRICE_PER_SQUARED_METER);
System.out.println("Price of lot 6 = " + area6 * PRICE_PER_SQUARED_METER);
System.out.println("Price of lot 7 = " + area7 * PRICE_PER_SQUARED_METER);
```

Maastricht University

Check Lecture2c.java before proceeding

Variables - Scope

- Local variables are available ("exist") during the execution of the block in which they are declared.

- Blocks are any lines of code that are surrounded by

```
{
    ...
}
```

Variables - Scope

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

    int k = 5;
    {
        int j = 0;        j lifetime

        j++;
        k++;
        System.out.println(j);
    }

    k++;
    System.out.println(k);

}
```

k (and args) lifetime

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

    int k = 5;
    {
        int j = 0;        compilation
        int k = 3;        error

        j++;
        k++;
        System.out.println(j);
    }

    k++;
    System.out.println(j);        compilation
                                  error
}
```

Variables - Garbage Collection

- When a variable's scope ends, it is still somewhere present in memory (RAM), even though it cannot be accessed anymore.

- The **Garbage Collector** will remove the value from memory for dynamically-allocated values.

Exercise together – How can we extend the lifetime of a variable (2d)

- Create one block of code where you calculate the area of a rectangle.
- Create another block of code where you print "The area of rectangle X by Y is Z square meters."
- How can we reuse the variables from the first block in the second block?

Assignment operator

= assigns (stores) the value of an expression (in)to a variable

<variable name> = <expression>;

The variable type has to match the expression type (although type widening is often automatic)

E.g.   int width, height;
       double area;
       width = 5;
       height = 3;
       area = width * height / 2.0;

Maastricht University

Variable instantiation / initialization

= assigning an initial value (during declaration)

e.g. boolean a = true;

int width = 5, height = 3;

boolean result = !a && (height < 5);

double area = width * height / 2.0;

Forgetting to initialize before use will lead to compilation error!

e.g. int x, y;

x = y*y;

leads to:

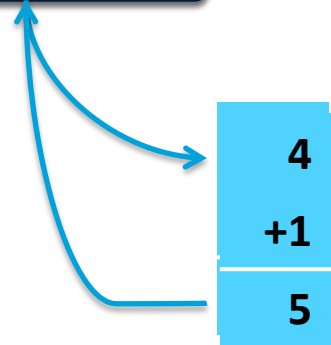Tmp.java:8: variable y might not have been initialized x = y*y;

Assignment ≠ Equality

int x;

x = 4;

x = x+1;

Shortcuts:
x++; is same as x = x+1;
x+=3; is same as x = x+3;
also for x--; and x-=3;

x ⟶ [   ]

x ⟶ [4   ]

x ⟶ [5   ]

4

+1

5

Exercise together – Swap values

- Let's write a program where we have two variables (say x which is 5 and y which is 10) and we want to swap their values.

- Sounds easy…?

Methods - Learning Goals

- You know how to declare methods
- You know how to group reusable statements into methods
- You know what a parameter is and how to use them
- You know how to return values from a method
- You know the terminology related to methods

Methods

- Groups related statements that re-occur together in a task

Methods have a
- **name**
- **input**   called   "*parameters*"
- **output**   called   "***return*** *value*"
- **body**

Methods - Declaration

Access Modifier
(learnt later)

Return Type

Method name

Parameters

```
public int sum(int a, int b) {
    return a + b;
}
…
int z = obj.sum(1, 5);
```

Body

Object
(learnt later)

Method
(call / invocation)

Maastricht University

```java
public class Example {

    public static void main(String[] args) {

        int a, b, c;
        a = 1;
        b = 2;
        c = 3;

        System.out.println(zrpt(a,b,c));
    }

    public static int zrpt(int x, int y, int z) {
        int sum = x + y + z;
        return sum;
    }
}
```

Check Lecture2e.java before proceeding

Things Java doesn't care about... but we might

```java
public class Example {

    public static void main(String[] args) {

        int a, b, c;
        a = 1;
        b = 2;
        c = 3;

        System.out.println(zrpt(a,b,c
    }

    public static int zrpt(int x, int
        int sum = x + y + z;
        return sum;

    }
}
```

1. Method name
2. Parameter names
3. Variable names

… as long as they match up!

```java
public class Example {

    public static void main(String[] args) {

        int a, b, c;
        a = 1;
        b = 2;
        c = 3;

        System.out.println(add(a,b,c));
    }

    public static int add(int p1, int second, int kurt) {
        int total = p1 + second + kurt;
        return total;
    }
}
```

Maastricht University

More things Java does not care about

4. Method order

```java
public class Example {

    public static void main(String[] args) {

        int a, b, c;
        a = 1;
        b = 2;
        c = 3;

        System.out.println(add(a,b,c));
    }

    public static int add(int x, int y, int z) {
        int sum = x + y + z;
        return sum;
    }
}
```

```java
public class Example {

    public static int add(int x, int y, int z) {
        int sum = x + y + z;
        return sum;
    }

    public static void main(String[] args) {

        int a, b, c;
        a = 1;
        b = 2;
        c = 3;

        System.out.println(add(a,b,c));
    }
}
```

Methods – Return Statement

- **return** <result_expression>;

- **Ends the execution** of the method and **returns a value**
- **Expression** type must match **return type** of method
- Mostly (only) occurs as last statement (compiler will complain if it does not)

- Special case: return type "void"
  - meaning: no return type
  - consequence: no return statement
  - can still use "return;" to end method execution; often considered bad form

Maastricht University

Method – Parameters

```
(<type1> <name1>, <type2> <name2>, … )
```

Input the method needs to compute the correct result

- comma separated list of variables with their type; declared between parentheses
- list can be empty, brackets are still required
- lifespan = duration of the method call

Maastricht University

Overloading

In Java, methods are identified by the combination of name and parameter types

Overloading is defining methods with the same name but different parameter counts or types

```java
public static int add(int x, int y, int z) {
    System.out.println("    In int adder");
    return x + y + z;
}

public static int add(double x, double y, double z) {
    System.out.println("    In double to int adder");
    return (int) (x + y + z);
}
```

Overloading (2)

The return type of a method is NOT part of its identity!!

```java
public static int add(double x, double y, double z) {
    System.out.println("In double to int adder");
    return (int) (x + y + z);
}

public static double add(double x, double y, double z) {
    System.out.println("In double adder");
    return x + y + z;
}
```

Compilation Error!

Calling a method

```
int result = add(3,4,5);
```

Calling or invoking a method executes the body of the method and allows collecting the results
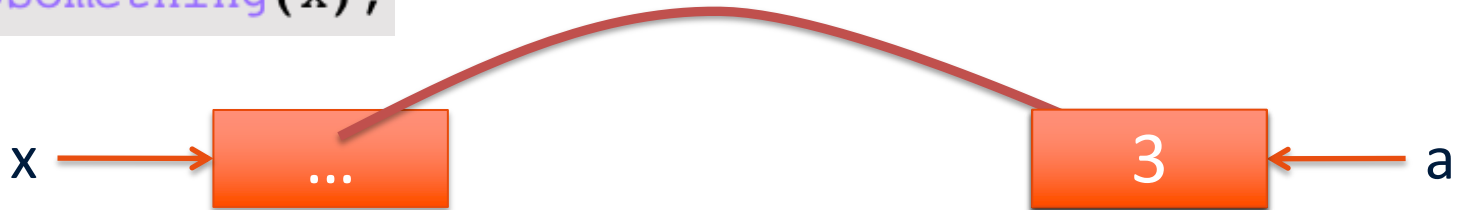- factual parameter values have to match the type of the method definition (or be able to become the correct type through widening)
- "pass-by-value": values are copied
- collecting the result is done by assigning the method expression to a suited variable

Maastricht University

Pass by value

```java
public static void doSomething(int a) {
    ...
    a = 3;
    ...
}
```

```java
int x = ...;
doSomething(x);
```

x → [...]   [3] ← a

Check Lecture2f.java before proceeding

Maastricht University

Show of hands

- What is printed in the output of this code?

- 3
- 44
- doSomething
- Nothing

```java
public class Lecture2f {
    Run | Debug
    public static void main(String[] args) {

        int x = 44;
        System.out.println(x);
        //call the method
        doSomething(x);
        System.out.println(x);
    }

    public static int doSomething(int x) {
        x = 3;
        return x;
    }
}
```

Local variables

Additional variables can be defined inside a methods body

- lifetime is the execution of the method
  (as for parameters)
  or the surrounding block as always

```java
public class MethodCaller
{

    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;
```

Trouble?

Nope!

Lifespans do not overlap: no connection between the two names!

Control Flow & Variable Scopes
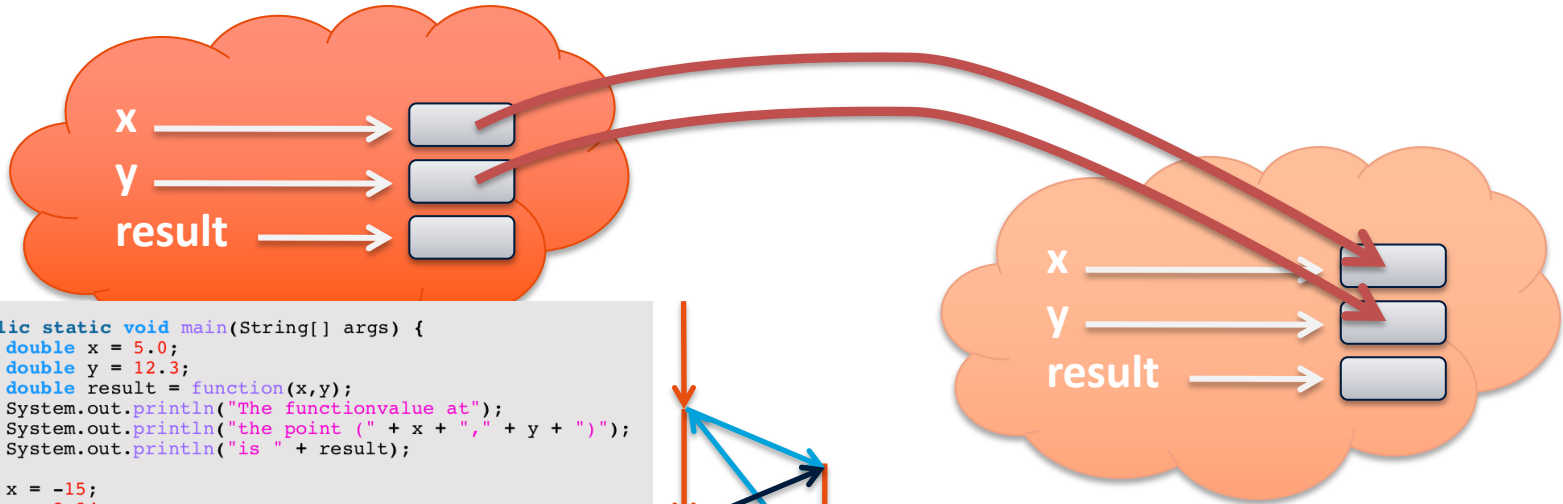

```java
public static void main(String[] args) {
    double x = 5.0;
    double y = 12.3;
    double result = function(x,y);
    System.out.println("The functionvalue at");
    System.out.println("the point (" + x + "," + y + ")");
    System.out.println("is " + result);

    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;
}
```

Maastricht University

Check MethodCaller.java before proceeding

Math Library

Math.<method>(<parameters>)

E.g.   Math.sqrt(x)
                Math.min(x,y)
                Math.exp(x)
                Math.pow(x,y)
                Math.cos(x)   (radian! – Math.toRadians(x))
                Math.round(x)

Look at Java API!!
https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/lang/Math.html

Maastricht University

Check FunWithMath.java before proceeding

Reading input

For most of this course: command line or terminal

In Java 5.0, Scanner was added to read keyboard input
- nextInt() reads an int
- nextDouble() reads a double
- nextLine() reads a line (until <enter>)
- next() reads a word (until any white space)

Scanner is not a "library" like Math, but requires
instantiation(=declare it, like we do with variables)

Using Scanner

```java
import java.util.Scanner;

public class ComplexHelloWorld {

    public static void main(String[] args) {

        //display a more complex greeting in the console window
        Scanner in = new Scanner(System.in);

        System.out.println("Please type your name and press enter.");
        String name = in.nextLine();
        System.out.println("Hello " + name);

        System.out.println("What is your age?");
        int age = in.nextInt();
        System.out.println("You will be " + (age+10) + " in ten years.");
    }

}
```

Check ComplexHelloWorld.java before proceeding

Summary

- Variables
  - declaration, initialization, lifespan
  - assignment, expressions
- Methods
  - definition and invocation, parameters

- **Book**: *Chapters 2 & 5 (check canvas for details)*
- **Quizzes**: *1b & 2*
- **Homework**: *Tasks 1 to 12*

Maastricht University

The Upcoming Practical

- This Friday!
  - Study the slides, book chapters, quiz 1b and 2, homework before the lab!

- On variables and methods
  - Make sure that you have reached all learning goals after this week.
  - Learning to program requires knowledge of the *basics*, on which you can build ever more complex structures.