3. Java - Language Constructs I

Names and Identifiers, Variables, Assignments, Constants, Datatypes, Operations, Evaluation of Expressions, Type Conversions

Educational Objectives

- You know the basic blocks of the programming language Java
- You understand the use of variables in a program and you can use them properly
- You know how values are defined in the source code (literals)
- You are able to read and interpret simple arithmetic expressions
- You understand the reasons for a type system and are able to determine the type of an expression

Definition: Names and Identifiers

Names denote things in a program like variables, constants, types, methods, or classes.

A program (that is, a class) needs a name

```
public class SudokuSolver { ...
```

Convention for class names: use CamelCase → Words are combined into one word, each starting with a capital letter

Allowed names for “entities” in a program:
- Names begin with a letter or _ or $
- Then, sequence of letters, numbers or _ or $

Book, on page 21
### Names - what is allowed

- `_myName`
- `TheCure`
- `__AN$WE4_1S_42__`
- `me@home`
- `49ers`
- `strictfp` ?!
- `side-swipe`
- `Ph.D’s`
- `$bling$

### Keywords

The following words are already used by the language and cannot be used as names:

- `abstract`  
- `continue`  
- `for`  
- `new`  
- `switch`  
- `assert`  
- `default`  
- `goto`  
- `package`  
- `synchronized`  
- `boolean`  
- `do`  
- `if`  
- `private`  
- `throw`  
- `break`  
- `double`  
- `implements`  
- `protected`  
- `throw`  
- `byte`  
- `else`  
- `import`  
- `public`  
- `transient`  
- `case`  
- `enum`  
- `instanceof`  
- `return`  
- `try`  
- `catch`  
- `extends`  
- `int`  
- `short`  
- `void`  
- `class`  
- `final`  
- `interface`  
- `static`  
- `while`  
- `const`  
- `float`  
- `native`  
- `super`  

### Definition: Variables

Variables are **buckets** for values and have a specified **type**. Variables need to be **declared** before first use.

*Book, on page 23*

### Variables

- Variables are **buckets** for a value
- Have a **data type** and a **name**
- The data type determines what kind of values are allowed in the variable

#### Declaration in Java:

```
int x = 23, y = 42;
float f;
char c = 'a';
```

#### Initialization:

```
23  42  0.0f  ’a’
```
**Definition: Constants**

Constants are variables that are initialized upon declaration and may not change their value later on.

Book, on page 35

---

**Constants**

- **Keyword final**
- The value of the variable can be set exactly once

**Example**

```java
final int maxSize = 100;
```

**Hint:** Always use `final`, unless the value actually needs to change over time.

---

**Definition: Types**

A Type defines a set of values that belong to the type as well as a set of operations that can be performed with the values of the type.

Book, on page 24

---

**Definition: Standard Types**

Java provides several predefined types for various numeric ranges as well as boolean values and strings.

Book, on page 24
### Standard Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Definition</th>
<th>Value Range</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>8-bit integer</td>
<td>$-128, \ldots, 127$</td>
<td>0</td>
</tr>
<tr>
<td>short</td>
<td>16-bit integer</td>
<td>$-32'768, \ldots, 32'767$</td>
<td>0</td>
</tr>
<tr>
<td>int</td>
<td>32-bit integer</td>
<td>$-2^{31}, \ldots, 2^{31} - 1$</td>
<td>0</td>
</tr>
<tr>
<td>long</td>
<td>64-bit integer</td>
<td>$-2^{63}, \ldots, 2^{63} - 1$</td>
<td>0L</td>
</tr>
<tr>
<td>float</td>
<td>32-bit floating point</td>
<td>$\pm1.4E^{-45}, \ldots, \pm3.4E^{+38}$</td>
<td>0.0f</td>
</tr>
<tr>
<td>double</td>
<td>64-bit floating point</td>
<td>$\pm4.9E^{-324}, \ldots, \pm1.7E^{+308}$</td>
<td>0.0d</td>
</tr>
<tr>
<td>boolean</td>
<td>logical value</td>
<td>true, false</td>
<td>false</td>
</tr>
<tr>
<td>char</td>
<td>unicode-16 character</td>
<td><code>'\u0000', \ldots, 'a', 'b', \ldots, '\uFFFF'</code></td>
<td><code>'\u0000'</code></td>
</tr>
<tr>
<td>String</td>
<td>string</td>
<td>$\infty$</td>
<td>null</td>
</tr>
</tbody>
</table>

### Definition: Literals

Representation of a value of a standard type in the source code.

Book, on page 22 - 23

### Types and Memory Usage

Reminder: Memory cells contain 1 Byte = 8 bit

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td><code>true, false</code></td>
</tr>
<tr>
<td>byte</td>
<td><code>12 : value 12</code></td>
</tr>
<tr>
<td></td>
<td><code>-3 : value -3</code></td>
</tr>
<tr>
<td>short, char</td>
<td><code>25_872_224L : value 25'872'224</code></td>
</tr>
</tbody>
</table>

Hint: Underscores between digits are allowed!
**Literals: Floating Point Numbers**

are different from integers by providing

- decimal comma
  
  1.23e-7f

- and / or exponent.
  
  1.23e-7 : type double, value 1.23 · 10^{-7}

**Literals: Characters and Strings**

- Individual characters:
  
  'a' : Type char, value 97

- Strings:
  
  "Hello There!" : Type String

  "a" : Type String

Mind: Characters and Strings are two different things!

**Character: In ASCII Table**

<table>
<thead>
<tr>
<th>Character</th>
<th>ASCII Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>65</td>
</tr>
<tr>
<td>B</td>
<td>66</td>
</tr>
<tr>
<td>C</td>
<td>67</td>
</tr>
<tr>
<td>D</td>
<td>68</td>
</tr>
<tr>
<td>E</td>
<td>69</td>
</tr>
<tr>
<td>F</td>
<td>70</td>
</tr>
<tr>
<td>G</td>
<td>71</td>
</tr>
<tr>
<td>H</td>
<td>72</td>
</tr>
<tr>
<td>I</td>
<td>73</td>
</tr>
<tr>
<td>J</td>
<td>74</td>
</tr>
<tr>
<td>K</td>
<td>75</td>
</tr>
<tr>
<td>L</td>
<td>76</td>
</tr>
<tr>
<td>M</td>
<td>77</td>
</tr>
<tr>
<td>N</td>
<td>78</td>
</tr>
<tr>
<td>O</td>
<td>79</td>
</tr>
<tr>
<td>P</td>
<td>80</td>
</tr>
<tr>
<td>Q</td>
<td>81</td>
</tr>
<tr>
<td>R</td>
<td>82</td>
</tr>
<tr>
<td>S</td>
<td>83</td>
</tr>
<tr>
<td>T</td>
<td>84</td>
</tr>
<tr>
<td>U</td>
<td>85</td>
</tr>
<tr>
<td>V</td>
<td>86</td>
</tr>
<tr>
<td>W</td>
<td>87</td>
</tr>
<tr>
<td>X</td>
<td>88</td>
</tr>
<tr>
<td>Y</td>
<td>89</td>
</tr>
<tr>
<td>Z</td>
<td>90</td>
</tr>
</tbody>
</table>

**Definition: Assignments**

An assignment is used to store a (computed) value into a variable.

Book, on page 27
Value Assignment

Copies a value into variable \( x \)

- In pseudo code: \( x \leftarrow \text{value} \)
- In Java: \( x = \text{value} \)

“\( = \)” is the assignment operator \textit{and not a comparison}!

Therefore, \texttt{int y = 42} is both a declaration + an assignment.

Definition: \textit{Arithmetic Expressions}

An arithmetic expression consists of operands and operators and computes a numeric value of a given type.

Book, on page 28

Value Assignment

Examples

\begin{verbatim}
int a = 3;
double b;
b = 3.141;
int c = a = 0;
String name = "Inf";
\end{verbatim}

A \textit{nested} assignment: The expression \( a = 0 \) stores the value 0 into variable \( a \). \textit{and then returns the value}

Arithmetic Binary Operators

Infix notation: \( x \text{ op } y \) with the following operators

\begin{align*}
\text{op:} & \quad + \quad - \quad * \quad / \quad \%
\end{align*}

- \textbf{Precedence}: Multiplication, division, and modulo first, then addition and subtraction
- \textbf{Associativity}: Evaluation from left to right
### Arithmetic Binary Operators

- Division $x / y$: Integer division if $x$ and $y$ are integer.
- Division $x / y$: Floating-point division if $x$ or $y$ is a floating-point number!

#### Examples

**Integer division and modulo**
- $5 / 3$ evaluates to $1$
- $-5 / 3$ evaluates to $-1$
- $5 \% 3$ evaluates to $2$
- $-5 \% 3$ evaluates to $-2$

### Arithmetic Assignment

$x = x + y$

**Examples:**
- $x -= 3$; // $x = x - 3$
- name += "x" // name = name + "x"
- num *= 2; // num = num * 2

Analogous for $-\, , *\, , /\, , \%$

### Arithmetic Unary Operators

Prefix notation: $+x$ or $-x$

**Precedence:** Unary operators bind stronger than binary operators

#### Examples

Assuming $x$ is $3$
- $2 * -x$ evaluates to $-6$
- $-x - +1$ evaluates to $-4$

### Increment/Decrement Operators

Increment operators $++x$ and $x++$ have the same effect: $x \leftarrow x + 1$. But different return values:

- **Prefix operator** $++x$ returns the new value:
  - $a = ++x$; $\iff x = x + 1; a = x$;
- **Postfix operator** $x++$ returns the old value:
  - $a = x++$; $\iff$ temp = $x$; $x = x + 1$; a = temp;

**Precedence:** Increment and decrement operators bind stronger than unary operators

Analogous for $-\, -\, , -\, -x$. 

### Increment/Decrement Operators

**Examples**

Assuming `x` is initially set to 2

- `y = ++x * 3` evaluates to: `x` is 3 and `y` is 9
- `y = x++ * 3` evaluates to: `x` is 3 and `y` is 6

### Expressions

- **Examples**
  - primary: 
    - 
    - composed:
  - The type of “12 * 2.1f” is float

### Celsius to Fahrenheit

```java
public class Main {
    public static void main(String[] args) {
        Out.print("Celsius: ");
        int celsius = In.readInt();
        float fahrenheit = 9 * celsius / 5 + 32;
        Out.println("Fahrenheit: "+fahrenheit);
    }
}
```

**Example:** 15° Celsius are 59° Fahrenheit
Celsius to Fahrenheit - Analysis

9 * celsius / 5 + 32

Rule 1: Precedence

Multiplicative operators (\(*\), \(/\), \(\%\)) have a higher precedence ("bind stronger") than additive operators (\(+\), \(-\)).

Example

9 * celsius / 5 + 32
means
(9 * celsius / 5) + 32

Rule 2: Associativity

Arithmetic operators (\(*\), \(/\), \(\%\), \(+\), \(-\)) are left-associative: in case of the same precedence, the evaluation happens from left to right.

Example

9 * celsius / 5 + 32
means
((9 * celsius) / 5) + 32

Rule 3: Arity

Unary operators +, - before binary operators +, -.

Example

9 * celsius / 5 + 32
means
9 * celsius / (+5) + 32
Bracketing

Any expression can be bracketed unambiguously using the

- associativities
- precedences
- arities (number of operands)

of the involved operators.

Expression Trees

Bracketing leads to an expression tree

(((9 * celsius) / 5) + 32)

Expression Trees – Notation

Usual notation: root on top

9 * celsius / 5 + 32

Evaluation Order

“From leafs to the root” in the expression tree

9 * celsius / 5 + 32

Expression Trees – Notation

Usual notation: root on top

9 * celsius / 5 + 32
**Definition: Type System**

A type system is a set of rules that are applied to the different constructs of the language.

Book, on page 24

---

**Type System**

Java features a *static* type system:

- All types must be declared
- If possible, the compiler checks the typing . . .
- . . . otherwise it’s checked at run-time

Advantages of a static type system

- *Fail-fast* Bugs in the program are often found already by the compiler
- Understandable code

---

**Type errors**

**Example**

```java
int pi_ish;
float pi = 3.14f;
pi_ish = pi;
```

Compiler error:

```
./Root/Main.java:12: error: incompatible types: possible lossy conversion from float to int
pi_ish = pi;
```

---

**Explicit Type Conversion**

**Example**

```java
int pi_ish;
float pi = 3.14f;
pi_ish = (int) pi;
```

Explicit type conversion using casts (type)

- Statically type-correct, compiler is happy
- Run-time behavior: depends on the situation
  *Here: loss of precision: 3.14 \(\Rightarrow\) 3*
- Can crash a program at run-time
Type Conversion - Visually for Integer Numbers

Potential loss of information when casting explicitly, because less memory available to represent the number

Definition: Mixed Expressions

A mixed expression consists of operands of different types.

Book, on page 70

Mixed Expressions, Conversion

- Floating point numbers are more general than integers.
- In mixed expressions integers are converted to floating point numbers.

9 * celsius / 5 + 32

Type Conversions for Binary Operations

Numeric operands in a binary operation are being converted according to the following rules:

- If both operands have the same type, no conversion will happen
- If one operand is double, the other operand is converted to double as well
- If one operand is float, the other operand is converted to float as well
- If one operand is long, the other operand is converted to long as well
- Otherwise: Both operands are being converted to int