Java Fundamentals

Lesson 4: Storing and Managing Local Data

Speaker: Nicolae Sîrbu
Alexandru Umaneț

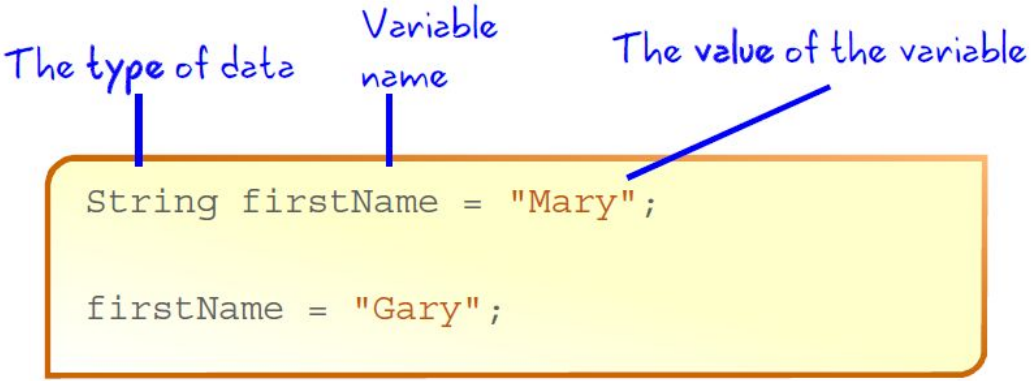
Lesson Objectives

- Introducing variable
- Primitive data types
- Working with `String` variables
- Working with numbers
- Manipulating numerical data

Introducing variable

Variables

- Each variable holds a specific **type** of data.
- The term variable refers to something that can change.
  - Variables can be initiated with a **value.**
  - The value can be changed.



Naming a variable

- Begin each variable with a lowercase letter. Subsequent words should be capitalized:
  - `myVariable, variable`
- Names are case-sensitive, `myVariable` and `myvariable` are two different variables.
- Names cannot include white spaces.
- Choose names that are mnemonic and that indicate to the casual observer, the intent of the variable:
  - `outOfStock` (a `boolean`)
  - `itemDescription` (a `String`)
  - `quantity` (a number)

Variables Declaration and Initialization

- Basic example:

```
String address = "123 Oak St";     //One variable declared
                                   //and initialized
```

type: String
identifier: address
value: "123 Oak St"

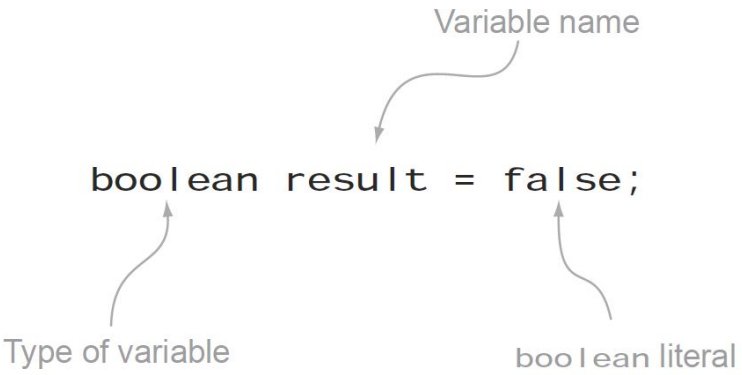
- Other examples:

```
String customer;                          //One variable declared

String name, city;                        //Two variables declared

String country ="USA", state ="CO";       //Two variables declared
                                          //and initialized
city = "Denver";                  //One variable initialized after
                                  //being declared earlier
```

Java - a strongly typed language

In some languages, such as JavaScript, you don't need to define the type of a variable before you use it. The compiler defines the type of the variable according to the value that you assign to it.

Java, in contrast, is a strongly typed language. You must declare a variable and define its type before you can assign a value to it.



Identifiers

Valid and Invalid Identifiers

Identifiers are names of packages, classes, interfaces, methods, and variables.

Properties of valid identifiers	Properties of invalid identifiers
Unlimited length	Same spelling as a Java reserved word or keyword
Starts with a letter (a–z, upper- or lowercase), a currency sign, or an underscore	Uses special characters: !, @, #, %, ^, &, *, (,), ', :, ;, [, /, \, }
Can use a digit (not at the starting position)	Starts with a Java digit (0–9)
Can use an underscore (at any position)	
Can use a currency sign (at any position): ¤, $, £, ¢, ¥, and others	

Valid and Invalid Identifiers

Examples of valid identifiers	Examples of invalid identifiers
`customerValueObject`	`7world` (identifier can't start with a digit)
`$rate, £Value, _sine`	`%value` (identifier can't use special char `%`)
`happy2Help, nullValue`	`Digital!, books@manning` (identifier can't use special char `!` or `@`)
`Constant`	`null, true, false, goto` (identifier can't have the same name as a Java keyword or reserved word)

Valid and Invalid Identifiers

```
int falsetrue;                    √
int javaseminar, javaSeminar;     √
int DATA-COUNT;                   X
int DATA_COUNT;                   √
int car.count;                    X
int %ctr;                         X
int ¥to£And$¢;                    √
```

Java Keywords and Reserved Words

Java keywords and reserved words that can't be used as names for Java variables

abstract	default	goto	package	this
assert	do	if	private	throw
boolean	double	implements	protected	throws
break	else	import	public	transient
byte	enum	instanceof	return	true
case	extends	int	short	try
catch	false	interface	static	void
char	final	long	strictfp	volatile
class	finally	native	super	while
const	float	new	switch	
continue	for	null	synchronized	

Primitive data types

Primitive data types

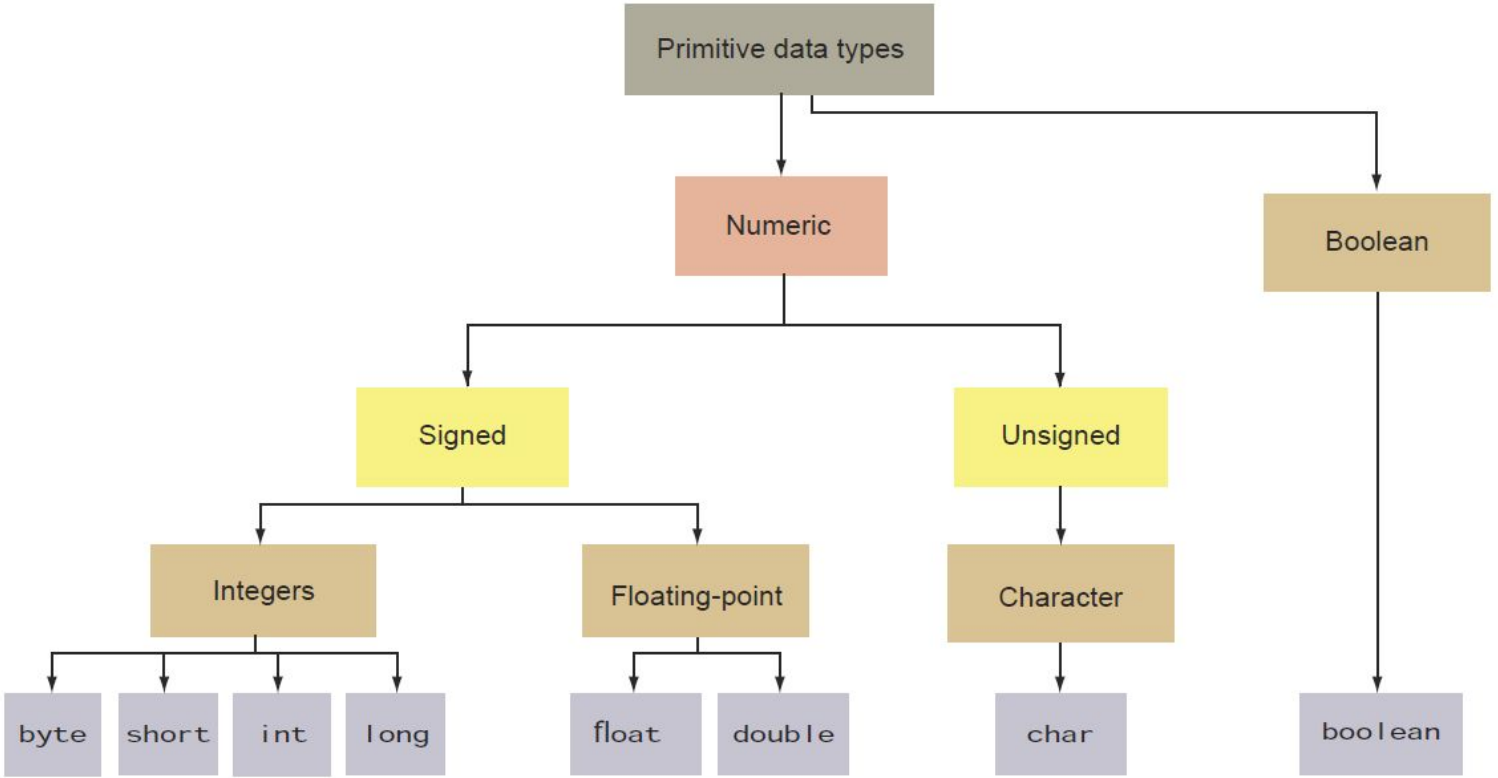
A variable defined as one of the primitive data types is a ***primitive variable***.

Primitive data types, as the name suggests, are the simplest data types in a programming language.

In the Java language, they're predefined:

- `char`
- `byte`
- `short`
- `int`
- `long`
- `float`
- `double`
- `boolean`

Primitive data types: Boolean



`boolean` data type

A `boolean` variable can store one of two values: `true` or `false`.

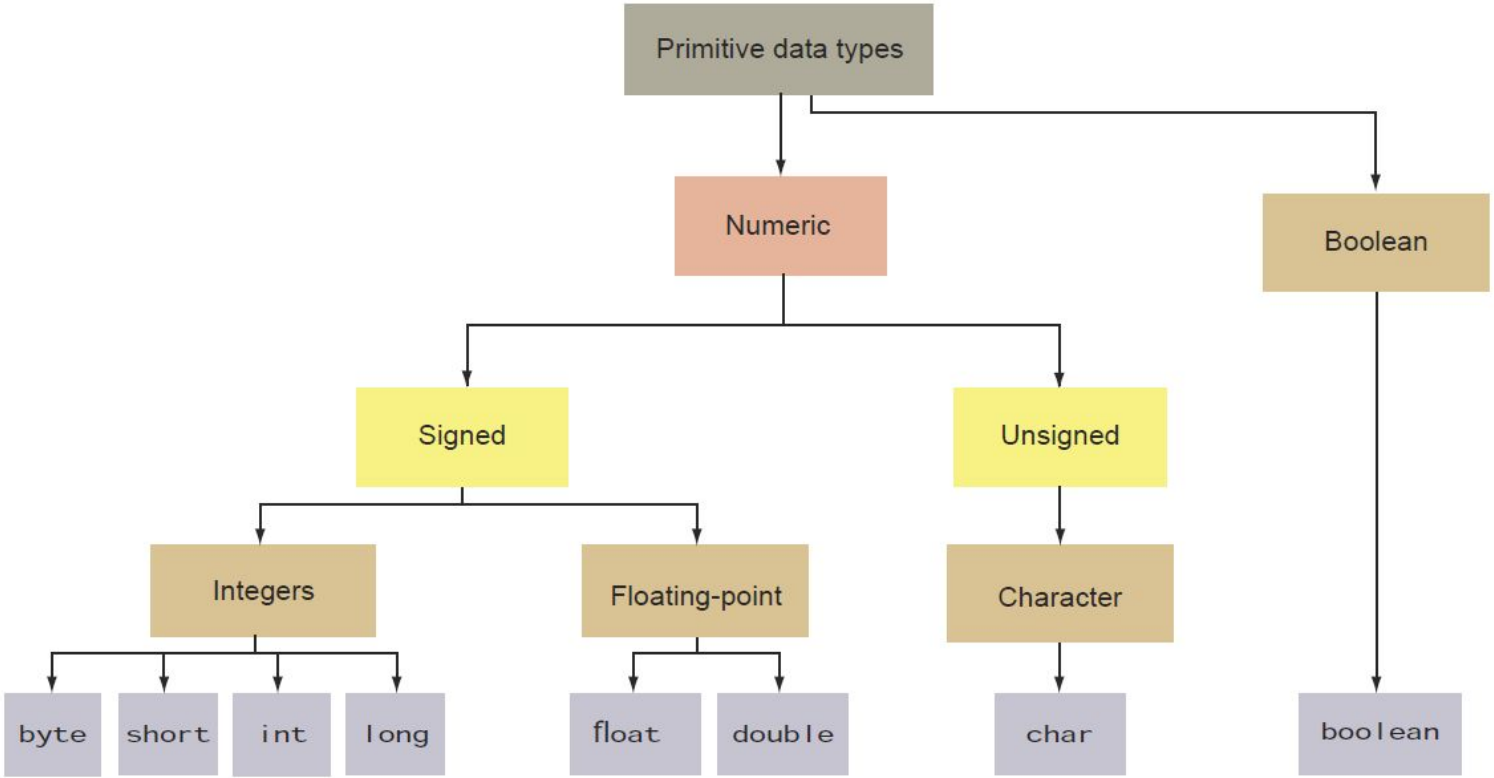
Here's some code that defines `boolean` primitive variables:

```
boolean voucherPurchased = true;
boolean examPrepStarted = false;
```

Exercise #4.1 Working with booleans

Write a Java program to declare a `boolean` variable with initial value of "`true`" and later change it to "`false`" before printing it.

Primitive data types: Signed numeric



Signed numeric data types: integers

The numeric category defines two subcategories: **integers** and **floating point** (also called **decimals**).

INTEGERS: `byte`, `short`, `int`, `long`

Type	Size in Bytes	Range
byte	1 byte	-128 to 127
short	2 bytes	-32,768 to 32,767
int	4 bytes	-2,147,483,648 to 2,147,483, 647
long	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Signed numeric data types: integers

Here's some code that assigns literal values to primitive numeric variables within their acceptable ranges:

```
byte num = 100;

short sum = 1240;

int total = 48764;

long population = 214748368;
```

Each of the integer types can store a different range of values. The benefits of the smaller ones are obvious: they need less space in memory and are faster to work with.

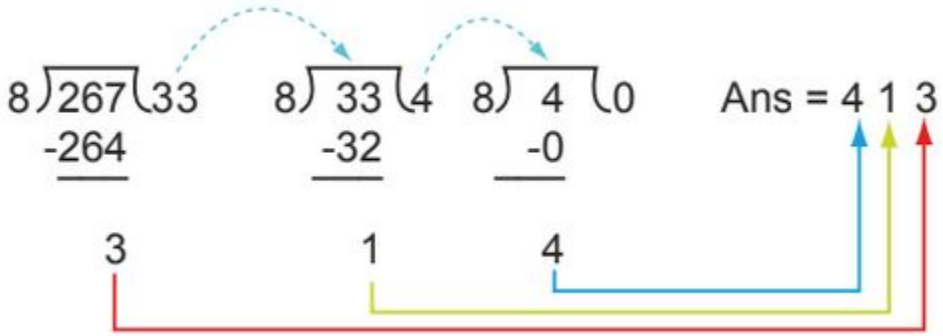
Number systems for an integer

Integer values come in four flavors: ***binary***, ***decimal***, ***octal***, and ***hexadecimal***:

- *Binary number system*: a base-2 system, which uses only 2 digits, 0 and 1.
- *Octal number system*: a base-8 system, which uses digits 0 through 7 (a total of 8 digits). Here the decimal number 8 is represented as octal 10, decimal 9 as 11,and so on.
- *Decimal number system*: the base-10 number system that you use every day. It's based on 10 digits, from 0 through 9 (a total of 10 digits).
- *Hexadecimal number system*: a base-16 system, which uses digits 0 through 9 andthe letters A through F (a total of 16 digits and letters). Here the number 10 is represented as A or a, 11 as B or b, 12 as C or c, 13 as D or d, 14 as E or e, and 15 as F or f.

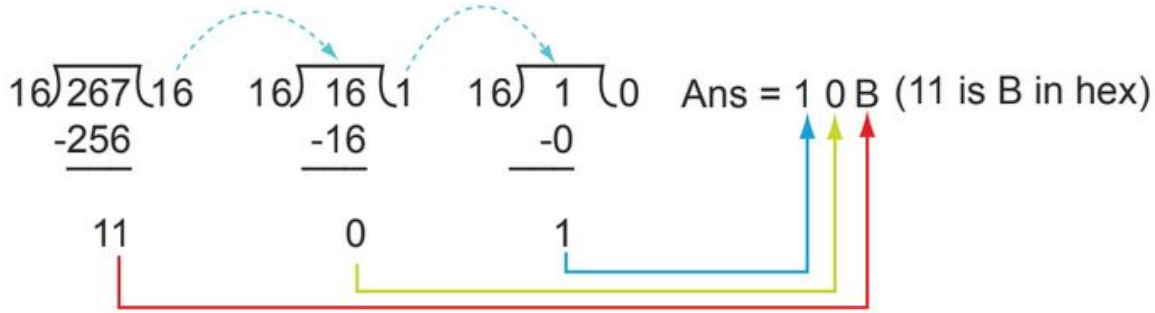
Converting an integer from decimal to octal

Converting the decimal number 267 to the octal number system:



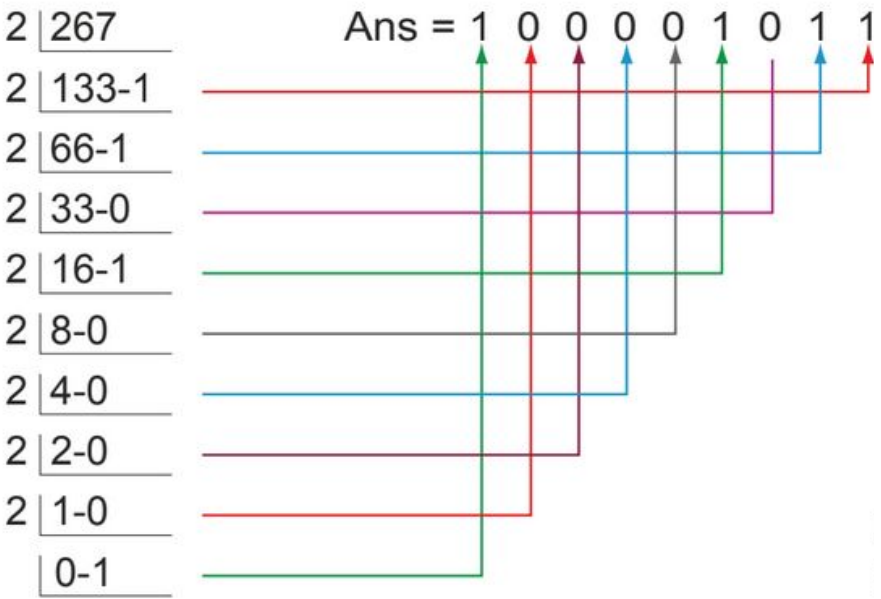
Converting from decimal to hexadecimal

Converting the decimal number 267 to the hexadecimal number system:



Converting an integer from decimal to binary

Converting the decimal number 267 to the binary number system:



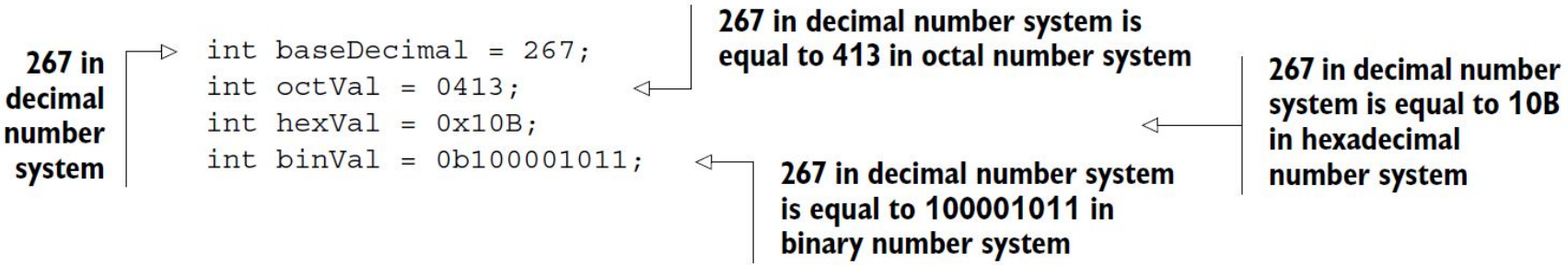
Integer initialization in all number systems

You can assign integer literals in base ***decimal***, ***binary***, ***octal***, and ***hexadecimal***.

For ***octal*** literals, use the prefix `0`.

For ***binary***, use the prefix `0B` or `0b`.

For ***hexadecimal***, use the prefix `0X` or `0x`.



More readable integer values

Java 7 introduced the use of underscores as part of the literal values. Grouping individual digits or letters of literal values makes them more readable.

- You can place an underscore right after the prefix `0`, which is used to define an octal literal value.
- You can't start or end a literal value with an underscore.
- You can't place an underscore right after the prefixes `0b`, `0B`, `0x`, and `0X`, which are used to define binary and hexadecimal literal values.
- You can't place an underscore prior to an `L` suffix (the `L` suffix is used to mark a literal value as long).
- You can't use an underscore in positions where a string of digits is expected.

More readable integer values

Correct:

- `long baseDecimal = 100_267_760;`
- `long octVal = 04_13;`
- `long hexVal = 0x10_BA_75;`
- `Long binVal = 0b1_0000_10_11;`

Wrong:

- `long var1 = 0_100_267_760;`
- `long var2 = 0_x_4_13;`
- `long var3 = _0x4_1;`
- `long var4 = 0b_10000_10_11;`
- `int i = Integer.parseInt("45_98");`

Signed numeric data types: integers

The default type of a non-decimal number is `int`. To designate an integer literal value as a `long` value, add the suffix `L` or `l` (`L` in lowercase), as follows:

```
long fishInSea = 764398609800L;
```

Signed numeric data types: decimals

The numeric category defines two subcategories: **integers** and **floating point** (also called **decimals**).

FLOATING-POINT NUMBERS: `float` and `double`

Type	Size in Bytes	Range
float	4 bytes	approximately ±3.40282347E+38F (6-7 significant decimal digits) *Java implements IEEE 754 standard*
double	8 bytes	approximately ±1.79769313486231570E+308 (15 significant decimal digits)

Signed numeric data types: decimals

Here's some code in action:

```
float average = 20.129F;
float orbit = 1765.65f;
double inclination = 120.1762d;
```

The default type of a decimal literal is `double`, but by suffixing a decimal literal value with `F` or `f`, you tell the compiler that the literal value should be treated like a `float` and not a `double`.

You can also add the suffix `D` or `d` to a decimal number value to specify that it's a `double` value. Because the default type of a decimal number is `double`, the use of the suffix `D` or `d` is redundant.

More readable decimal values

Starting with Java version 7, you can also use underscores with the floating-point literal values.

- You can't place an underscore prior to a `D`, `d`, `F`, or `f` suffix.
- You can't place an underscore adjacent to a decimal point.

Correct:

```
float pi = 3.14_15F;
double ssn = 99_999.9_999d;
```

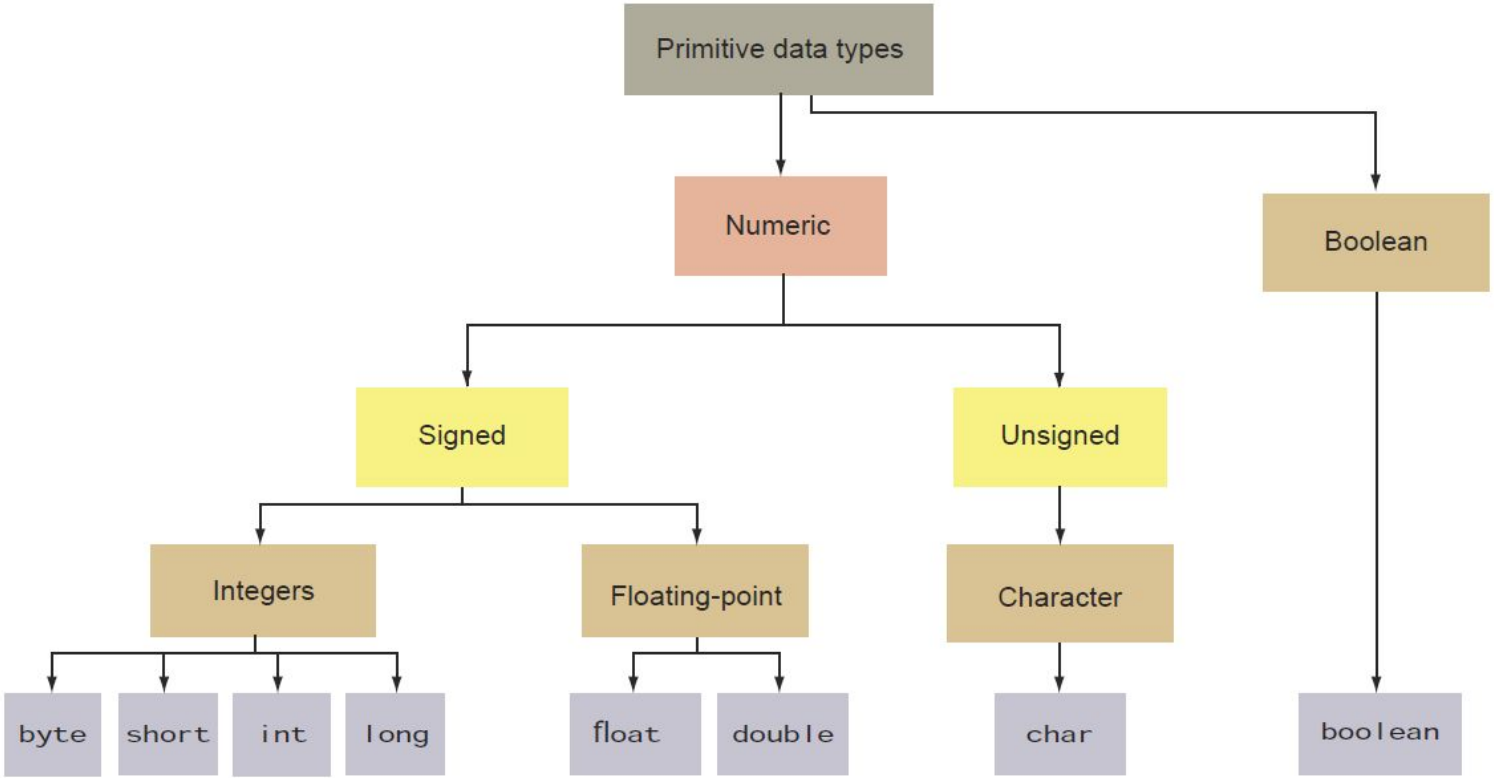
Wrong:

```
float pi = 3._14_15F;
double ssn = 99_999.9_999_d;
```

Exercise #4.2 Working with numbers

Write a Java program to declare three variable `a`, `b`, and `c` and store the values respectively `10`, `20.3` and `3.14785`. Then display their values on the screen.

Primitive data types: Unsigned numeric



Unsigned numeric data types

The character category defines only one data type: `char`.

```
char c1 = 'D';
char c1 = 68;
```

Internally, the `char` data type is stored as an unsigned `integer` value (only positive integers).

When you assign a letter to a `char`, Java stores its `integer` equivalent value.

* **Note**: Never use double quotes to assign a letter as a `char` value. Double quotes are used to assign a value to a variable of type `String`.

Unsigned numeric data types

`\u` is used to mark the value as a Unicode value. You must use quotes to assign Unicode values to char variables.

Here's an example:

```
char c1 = '\u0122';
System.out.println("c1 = " + c1);   // c1 = Ģ
```

Variables types default initialization

If uninitialized, the variables that are used as a class or instance variables get a default value:

- `byte, short, int, long -> 0`
- `float, double -> 0.0`
- `boolean -> false`

Uses of variables

- Holding data used within a method.
- Assigning the value of one variable to another.
- Representing values within a mathematical expression.
- Printing the value to the screen.

Confusion with the names of the primitive data types

If you've previously worked in another programming language, there's a good chance that you might get confused with the names of the primitive data types in Java and other languages.

```
public class MyProgram {
     public static void main(String[] args) {
          int myInt = 7;
          float myFloat = 5.5f;
          bool result = true;

          System.out.println("Value of myInt variable is " + myInt);
          System.out.println("Value of myFloat variable is " + myFloat);
          System.out.println("Value of result variable is " + result);

     }
}
```

Working with `String` variables

`String` concatenation

`String` variables can be combined using the '+' operator:

```
String greet1 = "Hello";
String greet2 = "World";
String message = greet1 + " " + greet2 + "!";
String message = greet1 + " " + greet2 + " " + 2016 +"!";
```

`String` concatenation output

You can concatenate `String` variables outside or inside a method call:

```
String greet1 = "Hello";
String greet2 = "World";
String message = greet1 + " " +greet2  + "!";

System.out.println(message);
System.out.println(greet1 + " " + greet2 + "!");
```

Output:

```
Hello World!
Hello World!
```

Exercise #4.3: Using `String` variables

1. Create a new project.
2. Create a Java class. Make sure the `main` class is created as well.
3. Declare and initialize two `String` variables inside the `main` method: `name` and `surname`, e.g:

```
String name = "Nicolae";
```

4. Declare a `String` variable called `message`. Do not initialize it.
5. Assign the `message` variable with a concatenation of the `name` and `surname`. Include a `String` literal that results in a complete sentence, e.g.: "My name is Nicolae Sirbu.".
6. Print the `message` to the System output.

Manipulating numerical data

Operator types and relevant operators

Operator type	Operators	Purpose
Assignment	`=, +=, -=, *=, /=`	Assign value to a variable
Arithmetic	`+, -, *, /, %, ++, --`	Add, subtract, multiply, divide, and modulus primitives
Relational	`<, <=, >, >=, ==, !=`	Compare primitives
Logical	`!, &&, ||`	Apply NOT, AND, and OR logic to primitives

Assignment operators

The simple assignment operator `=` is the most frequently used operator. It's used to initialize variables with values and to reassign new values to them.

The `+=`, `-=`, `*=`, and `/=` operators are short forms of addition, subtraction, multiplication, and division with assignment.

The `+=` operator can be read as "first add and then assign," and `-=` can be read as "first subtract and then assign."

Similarly, `*=` can be read as "first multiply and then assign," `/=` can be read as "first divide and then assign," and `%=` can be read as "first modulus and then assign."

Usage of assignment operators

`a -= b` is equal to `a = a - b`

`a += b` is equal to `a = a + b`

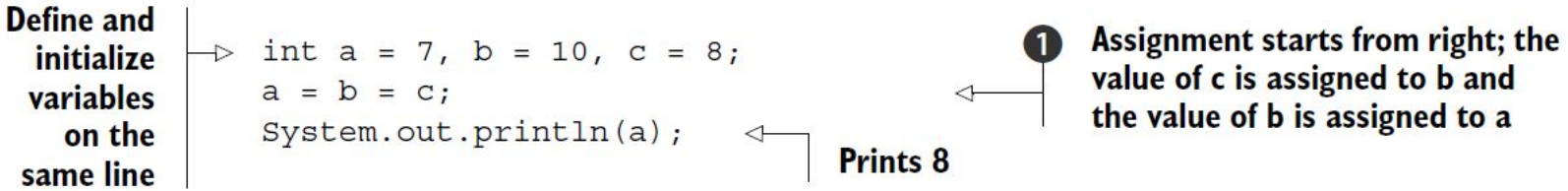
`a *= b` is equal to `a = a * b`

`a /= b` is equal to `a = a / b`

`a %= b` is equal to `a = a % b`

Usage of assignment operators

You can also assign multiple values on the same line using the assignment operator.



Arithmetic operators

Operator	Purpose	Usage	Answer
+	Addition	`12 + 10`	`22`
-	Subtraction	`19 - 29`	`-10`
*	Multiplication	`101 * 45`	`4545`
/	Division (quotient)	`10 / 6` `10.0 / 6.0`	`1` `1.6666666666666667`
%	Modulus (remainder in division)	`10 % 6` `10.0 % 6.0`	`4` `4.0`
++	Unary increment operator; increments value by 1	`++var` or `var++`	`11` (assuming value of `var` is 10)
--	Unary decrement operator; decrements value by 1	`--var` or `var--`	`9` (assuming value of `var` is 10)

Widening of data types in an arithmetic ops.

For arithmetic operations with data types `char`, `byte`, `short`, or `int`, all operand values are widened to `int`.

If an arithmetic operation includes the data type `long`, all operand values are widened to `long`.

If an arithmetic operation includes a data type of `float` or `double`, all operand values are widened to `double`.

```
byte age1 = 10;
byte age2 = 20;
short sum = age1 + age2;
```

Fails to compile

```
final byte age1 = 10;
final byte age2 = 20;
short sum = age1 + age2;
```

Compiles successfully

Widening primitive conversions

When we need to convert from a primitive that is smaller than the destination type, we don't have to use any special notation for that:

```
int myInt = 127;
long myLong = myInt;
```

During widening conversion, the smaller primitive value is placed over a larger container, which means that all the extra space, on the left of the value, is filled with zeros. This may also be used to go from the *integer* group to the *floating point*:

```
float myFloat = myLong;
double myDouble = myLong;
```

This is possible because the moving to a wider primitive does not lose any information.

Narrowing primitive conversion

Sometimes we need to fit a value that is larger than the type used in the variable declaration. This may result in information loss since some bytes will have to be discarded.

In this case, we have to explicitly express that we are aware of the situation and we agree with that, by using a cast:

```
int myInt = (int) myDouble;
byte myByte = (byte) myInt;
```

Exercise #4.4 Narrowing primitives

1. Declare a `long`, using the `L` to indicate a long value. Make it a very large number (billions).
2. Declare and initialize a `float`.
3. Print the `long` and the `float` variables with a suitable label.
4. Assign the `long` variable to an `int` variable. Correct the syntax error by casting the `long` as an `int`.
5. Assign the `float` variable to a `double` variable.
6. Print the `int` and double variables. Note the change in the values when you run it.

Exercise #4.5: Using and manipulating numbers

1. Declare and initialize numeric fields: `price` (double), `tax`(double) and `quantity`(int). Also declare `total`, but do not initialize it.
2. Create a `String` message to include `quantity`, e.g.: “I want to buy 1 shirt!” and print it.
3. Calculate `total` by multiplying `price * quantity * tax`
4. Print a message showing the total cost, e.g.: “Total cost with tax is: 29.6 ”.

Scanner Class in Java

Scanner is a class in `java.util package`used for obtaining the input of the primitive types like `int`, `double` etc. and `Strings`.

It is the easiest way to read input in a Java program.

```
Scanner sc = new Scanner(System.in);

String name = sc.nextLine();
int age = sc.nextInt();
long mobileNo = sc.nextLong();
double salary = sc.nextDouble();
```

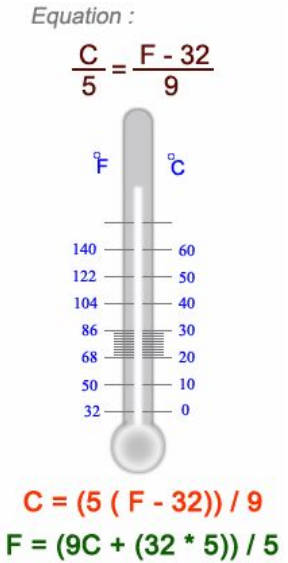
HM. Exercise #4.6: Fahrenheit to Celsius degree

Write a Java program to convert temperature from Fahrenheit to Celsius degree.

Ask the user to input the temperature.

Print the result to the console.

Add the program to your GitHub repository.



* Use the `Scanner` class to get the input of the user.

HM. Exercise #4.7: minutes to nr. of years and days

Write a Java program to convert minutes into:

1. a number of years;
2. a number of days.

Ask the user to input the minutes.

Print the result to the console.

Add the program to your GitHub repository.

* Use the `Scanner` class to get the input of the user.

HM. Exercise #4.8: Sum, difference, product, average

Write a Java program that accepts two integers from the user and then prints:

1. the addition
2. the subtraction
3. multiplication
4. the division
5. the average
6. the remainder

* Use the `Scanner` class to get the input of the user.

Increment & Decrement operators (++ and --)

The operators ++ and -- are ***unary operators***; they work with a single operand.

They're used to increment or decrement the value of a variable by 1.

The long way:

```
age = age + 1
```
or
```
count = count - 1
```

The short way:

```
age += 1
```
or
```
count -= 1
```

The shortest way:

```
age++
```
or
```
count--
```

Increment & Decrement operators (++ and --)

Operators can be used in ***prefix*** and ***postfix*** notation.

In ***prefix*** *notation*, the operator appears before its operand:

```
int a = 10;
++a;
```

Operator ++ in prefix notation

In ***postfix*** *notation*, the operator appears after its operand:

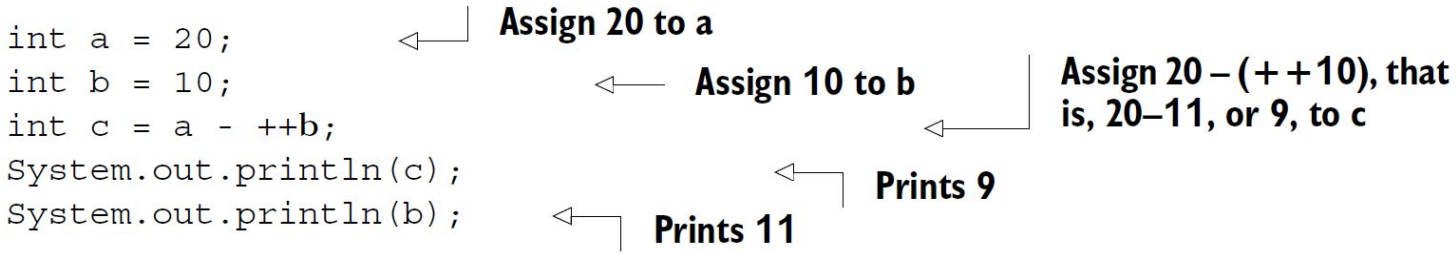
```
int a = 10;
a++;
```

Operator ++ in postfix notation

Increment & Decrement operators (++ and --)

`++a` increments and then uses the variable.

`a++` uses and then increments the variable.



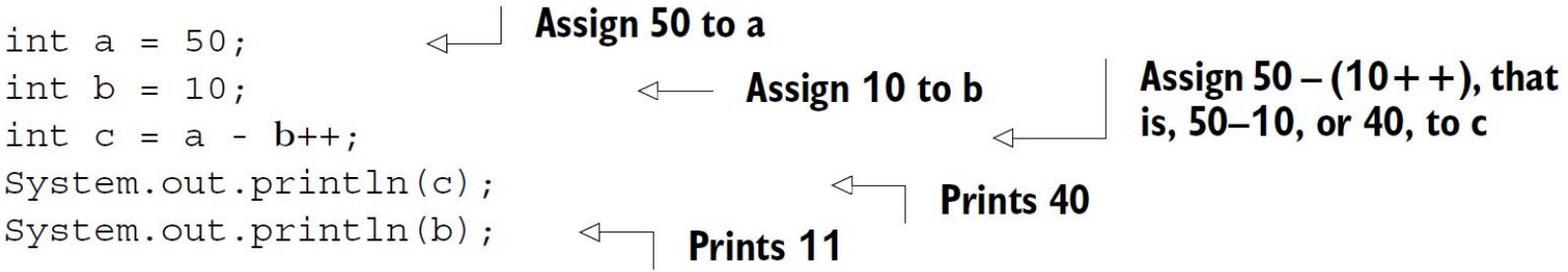
The interesting part here is that the value of `b` is printed as 11 in because the value of the variable increments (or decrements) as soon as the expression in which it's used is evaluated.

Exercise #4.9 Unary operators

What would be the output after the execution of the following code:

```
int a = 50;
int b = 10;
int c = a - b++;
System.out.println(c);
System.out.println(b);
```

Increment & Decrement operators (++ and −−)



Exercise #4.10 Unary operators

What would be the output after the execution of the following code:

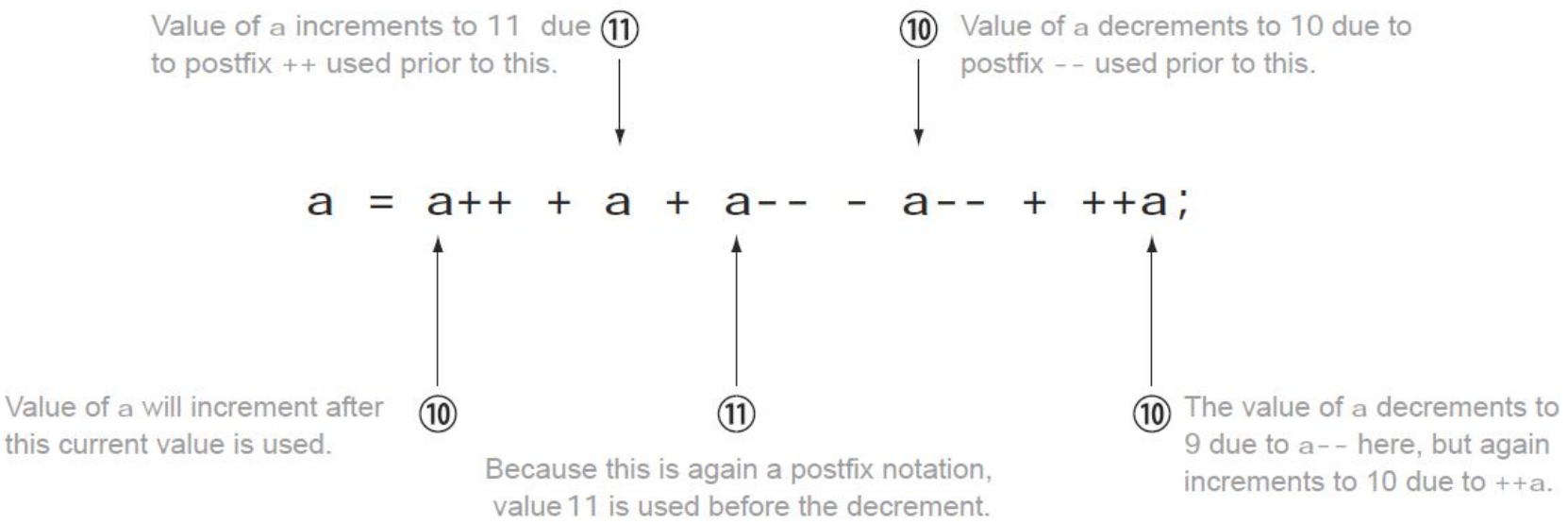
```
int a = 10;
a = a++ + a + a-- - a-- + ++a;
System.out.println(a);
```

The output of this code is 32.

a = 10 + 11 + 11 - 10 + 10;

Exercise #4.10 Explanation

Initial value of a is 10.



Exercise #4.11 Unary operators

Let's modify the expression used in the example below by replacing all occurrences of unary operators in ***prefix*** notation with ***postfix*** notations and vice versa.

So `++a` changes to `a++`, and vice versa. Similarly, `--a` changes to `a--`, and vice versa. Your task is to evaluate the modified expression and determine the output of the following code:

```
a = a++ + a + a-- - a-- + ++a;    // a = ?
a = ++a + a + --a - --a + a++;    // a = ?
a = a-- + a + a++ - a++ + --a;    // a = ?
a = --a + a + ++a - ++a + a--;    // a = ?
```

when the initial value of `a` before evaluating an expression is 10.

Relational operators

Relational operators are used to check one condition.

You can use these operators to determine whether a primitive value is equal to another value or whether it is less than or greater than the other value.

These relational operators can be divided into two categories:

- Comparing greater (>, >=) and lesser values (<, <=)
- Comparing values for equality (==) and inequality (! =)

Comparing primitives for equality (== and !=)

```
int a = 10;
int b = 20;
System.out.println(a == b);                 // false
System.out.println(a != b);                 // true

boolean b1 = false;
System.out.println(b1 == true);             // false
System.out.println(b1 != true);             // true
System.out.println(b1 == false);            // true
System.out.println(b1 != false);            // false
```

Comparing primitives using the assignment operator (=)

It's a very common mistake to use the assignment operator, =, in place of the equality operator, ==, to compare primitive values.

```
int a = 10;
int b = 20;
System.out.println(a = b);
boolean b1 = false;
System.out.println(b1 = true);
System.out.println(b1 = false);
```

1 Prints 20 (this is not a boolean value!)

2 Prints true

Prints false

Logical operators (`&&`, `||`, `!`)

Logical operators are used to evaluate one or more expressions.

These expressions should return a `boolean` value.

You can use the logical operators **AND (`&&`)**, **OR (`||`)**, and **NOT (`!`)** to check multiple conditions and proceed accordingly.

```
int a = 10;
int b = 20;
System.out.println(a > 20 && b > 10);       // false
System.out.println(a > 20 || b > 10);       // true
System.out.println(! (b > 10));             // false
System.out.println(! (a > 20));             // true
```

Outcome of using `booleans` with logical op.

Operators && (AND)	Operator || (OR)	Operator ! (NOT)

`true && true → true`
`true && false → false`
`false && true → false`
`false && false → false`
`true && true && false → false`
	`true || true → true`
	`true || false → true`
	`false || true → true`
	`false || false → false`
	`false || false || true → true`
		`!true → false`
		`!false → true`

`&&` and `||` are short-circuit operators

The `&&` operator returns `true` only if both the operands are `true`.

If the first operand to this operator evaluates to `false`, the result can never be `true`.

Therefore, `&&` does not evaluate the second operand.

Similarly, the `||` operator does not evaluate the second operator if the first operand evaluates to `true`.

```
int marks = 8;
int total = 10;
System.out.println(total < marks && ++marks > 5);  // false
System.out.println(marks);                         // 8
System.out.println(total != 10 || ++marks < 10);   // true
System.out.println(marks);                         // 9
```

Exercise #4.12 Logical operators

Examine the following code and determine the expressions that you think will evaluate.

```
int a = 10;
int b = 20;
int c = 40;

System.out.println(a++ > 10 || ++b < 30);                    // true

System.out.println(a > 90 && ++b < 30);                      // false

System.out.println(!(c>20) && a==10 );                       // false

System.out.println(a >= 99 || a <= 33 && b == 10);           // false

System.out.println(a >= 99 && a <= 33 || b == 10);           // false
```

Operator precedence

Is the answer in the following problem 34 or 9?

```
int c = 25 - 5 * 4 / 2 - 10 + 4;
```

Operator precedence

Operator	Precedence
Postfix	`Expression++, expression--`
Unary	`++expression, --expression, +expression, -expression, !`
Multiplication	`*` (multiply), `/` (divide), `%` (remainder)
Addition	`+` (add), `-` (subtract)
Relational	`<, >, <=, >=`
Equality	`==, !=`
Logical AND	`&&`
Logical OR	`||`
Assignment	`=, +=, -=, *=, /=, %=`

Changing operator precedence using parentheses

Example:

```
int c = (((25 - 5) * 4) / (2 - 10)) + 4;
int c = ((20 * 4) / (2 - 10)) + 4;
int c = (80 / (2 - 10)) + 4;
int c = (80 / -8) + 4;
int c = -10 + 4;
int c = -6;
```

Exercise #4.15 Working with data types

Which three options are correct in terms of Data Types and their syntax:

```
int value = "ToolsQA";                    // 1
int value = 67;                           // 2   √
boolean booleanValue = 67;                // 3
char charValue = 45;                      // 4   √
boolean booleanValue = "true";            // 5
boolean booleanValue = false;             // 6   √
```

Exercise #4.16 Syntax errors

Find the syntax error in the below program:

```
public static void main(String[] args) {
     System.out.println("ToolsQA");
}
```

Exercise #4.17 Syntax errors

Find the syntax error in the below program:

```
public static void main(String[] args) {
    System.out.println("Tools QA"));;
}
```

Exercise #4.18 Program errors

Find the error in the below program:

```
public static void main(String[] args) {
    int value = 18;
    System.out.println("The value of the integer variable is : " + value);
}
```

Exercise #4.19 Program errors

Find the error in the below program:

```
public static void main(String[] args) {
    int valueInt = 100;
    double valueDbl = 10.10;
    boolean valueBool = true;
}
```

Resources

Data Types in Java
(https://beginnersbook.com/2017/08/data-types-in-java/)

Java Operators
(https://www.geeksforgeeks.org/operators-in-java/)
(https://data-flair.training/blogs/java-operators/)

Resources

Head First Java, 2nd Edition
(http://www.proqramci.az/assets/book/head-first-java.pdf)

Java The Complete Reference, 9th Edition
(https://mitseu.files.wordpress.com/2014/08/java-the-complete-reference-ninth-editiona4.pdf)

Effective Java™ Second Edition
(https://the-eye.eu/public/Books/IT%20Various/Effective%20Java%2C%202nd%20Edition.pdf)

OCA Oracle Certified Associate Java SE 8 Programmer I Study Guide Exam 1Z0-808
(https://github.com/gopinathankm/Java-Training-2018/blob/master/OCA-Oracle%20Certified%20Associate%20Java%20SE%208%20Programmer%20I%20Study%20Guide%20Exam%201Z0-808.pdf)

Java Fundamentals

Lesson 4: Storing and Managing Local Data

End.

Speaker: Nicolae Sîrbu
Alexandru Umaneț