BBM 102 – Introduction to Programming II

Spring 2020

Introduction to Java

Instructors: *Sevil ŞEN, Selman BOZKIR, Cemil ZALLUHOĞLU*
TAs: *Nebi YILMAZ, Bahar GEZİCİ*

Today

Introduction

- About the class
- Organisation of this course

Introduction to Java

- Java as a Platform
- Your First Java Program
- Basic Programming Elements

Today

Introduction

- About the class
- Organisation of this course

Introduction to Java

- Java as a Platform
- Your First Java Program
- Basic Programming Elements

About the course

This course will help students understand object-oriented programming principles and apply them in the construction of Java programs.

- The course is structured around basic topics such as *classes, objects, encapsulation, inheritance, polymorphism, abstract classes and interfaces and exception handling*.

BBM 104 Introduction to Programming Practicum: The students will gain hand-on experience via a set of programming assignments supplied as complementary.

Requirements: You must know basic programming (i.e. BBM101).

BBM 102-104 Team

Instructors

Cemil ZALLUHOĞLU
(Section 1)

Selman BOZKIR
(Section 2)

Sevil ŞEN
(Section 3)

TAs

Nebi YILMAZ

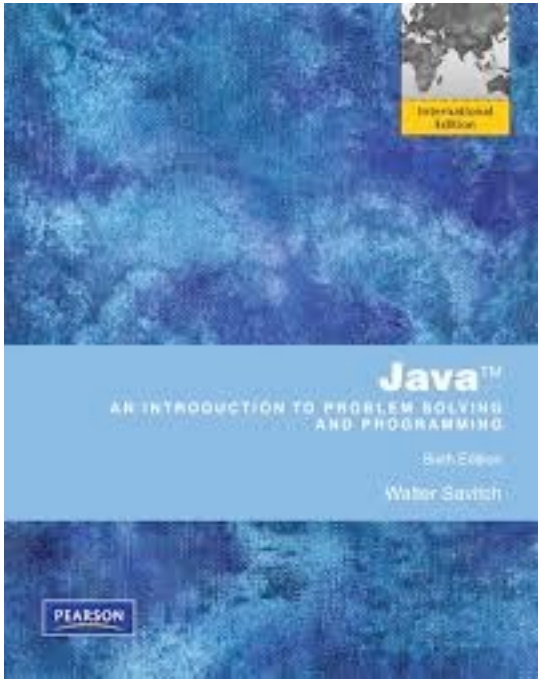
Bahar GEZİCİ

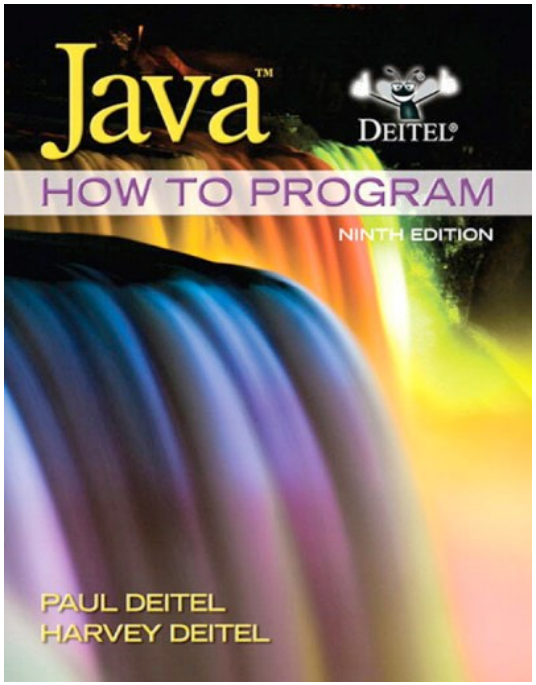
Office hours: See Web page

Reference Book

Java - An Introduction to Problem Solving and Programming, Walter Savitch, Pearson, 2012

Java - How to Program, Paul Deitel and Harvey Deitel, Prentice Hall, 2012





Communication

The course web page will be updated regularly throughout the semester with lecture notes, programming assignments, announcements and important deadlines.

http://web.cs.hacettepe.edu.tr/~bbm102

Getting Help

Office hours

- See the web page for details

BBM 104 Introduction to Programming Practicum

- Course related recitations, practice with example codes, etc.

Communication

- Announcements and course related discussions through piazza

BBM 102: https://piazza.com/hacettepe.edu.tr/spring2020/bbm102

BBM 104: https://piazza.com/hacettepe.edu.tr/spring2020/bbm104

Course Work and Grading

2 midterm exams (30 + 30 = 60%)

- Closed book and notes
- On week 6 (April 1st)and week 11 (May 6th), respectively.

Final exam (40%)

- Closed book
- To be scheduled by the registrar

The joy of learning

Class Attendance

- Attempting to create false attendance (e.g., signing in the attendance list on behalf of someone else) will be punished.
- If a student does not attend 4 courses during the semester, that student will fail because of absenteeism

Course Overview

Week	Date	Title
1	26-Feb	Introduction to Java
2	4-Feb	Object-Oriented Design
3	11-Mar	Classes and Objects in Java
4	18-Mar	Encapsulation
5	25-Mar	Inheritance
6	1-Apr	Review (Midterm Exam 1)
7	8-Apr	Polymorphism
8	15-Apr	Exceptions
9	22-Apr	Collections
10	29-Apr	Generics
11	6-May	Review (Midterm Exam 2)
12	13-May	Data Structures (Stack, Queue, Priority Queue)
13	20-May	Data Structures (Sets, Maps)
14	27-May	Algorithmic complexity

BBM 104 Introduction to Programming Practicum

Programming assignments (PAs)

- Four assignments throughout the semester.
- Each assignment has a well-defined goal such as solving a specific problem.
- You **must work alone** on all assignments stated unless otherwise.

Quizes

- Five quizes throughout this semester
- No extension

On Lab Assignment

- One lab assignment (this will be done in the lab)

Important Dates

- See the course web page for schedule.

Policies

Work groups

- You must work alone on all assignments stated unless otherwise

Submission

- Assignments due at 23:59 (no extensions!)
- Electronic submissions (no exceptions!)

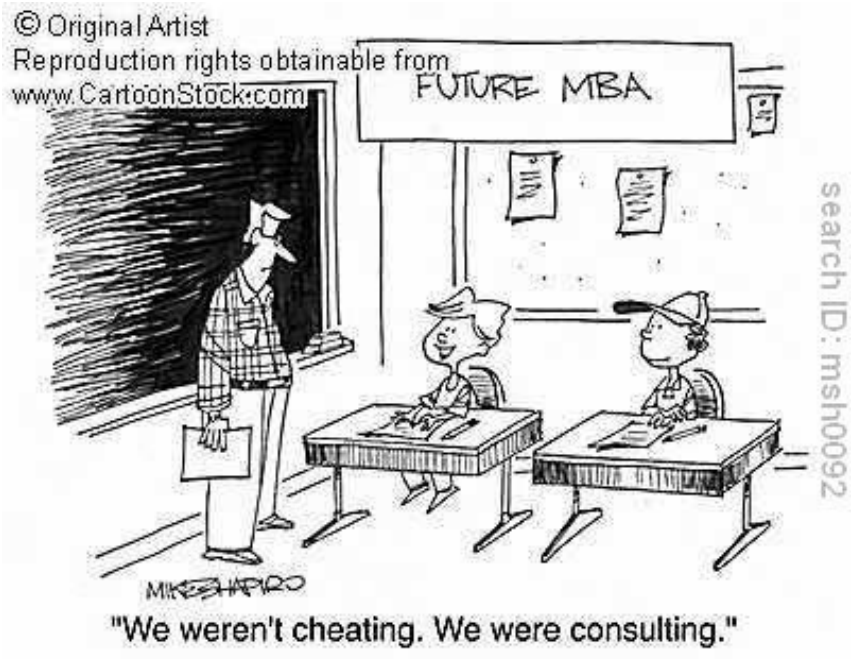
Lateness penalties

- Get penalised **10% per day**
- No late submission is accepted **3 days after due date**

Cheating

What is cheating?

- Sharing code: by copying, retyping, looking at, or supplying a file
- Coaching: helping your friend to write a programming assignment, **line by line**
- Copying code from previous course or from elsewhere on **WWW**



What is NOT cheating?

- Explaining how to use systems or tools
- Helping others with high-level design issues

Cheating

Penalty for cheating:

- Removal from course with failing grade

Detection of cheating:

- We do check: Our tools for doing this are much better than most cheaters think!

Today

Introduction

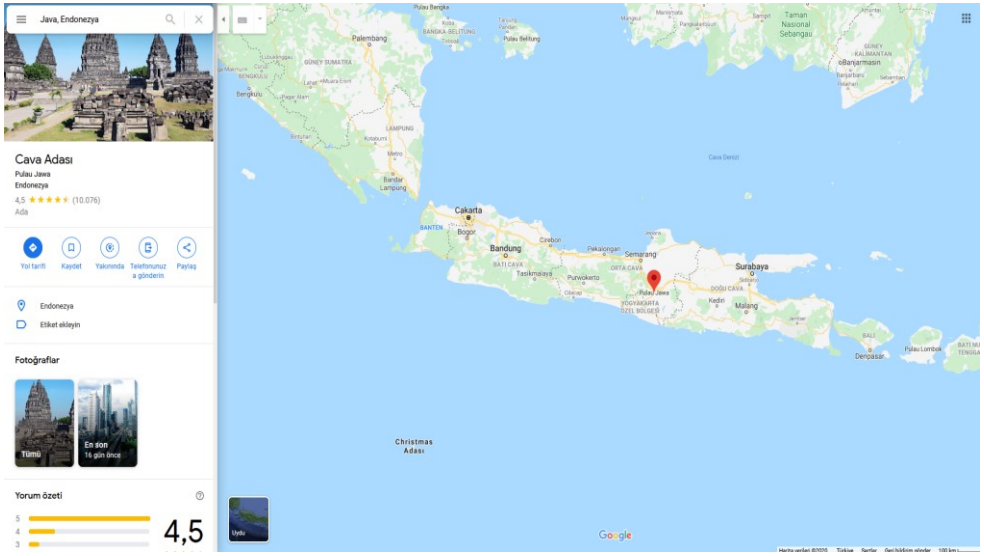
- About the class
- Organization of this course

Introduction to Java

- Java as a Platform
- Your First Java Program
- Basic Programming Elements

What is Java?

*An **island** of Indonesia lying between the Indian Ocean and the Java Sea.*



What is Java?

*Informal. Brewed **coffee**.*

What is Java?

A technology which is both a programming language and a platform.

Developed by Sun Microsystems.

First public version was released in 1995

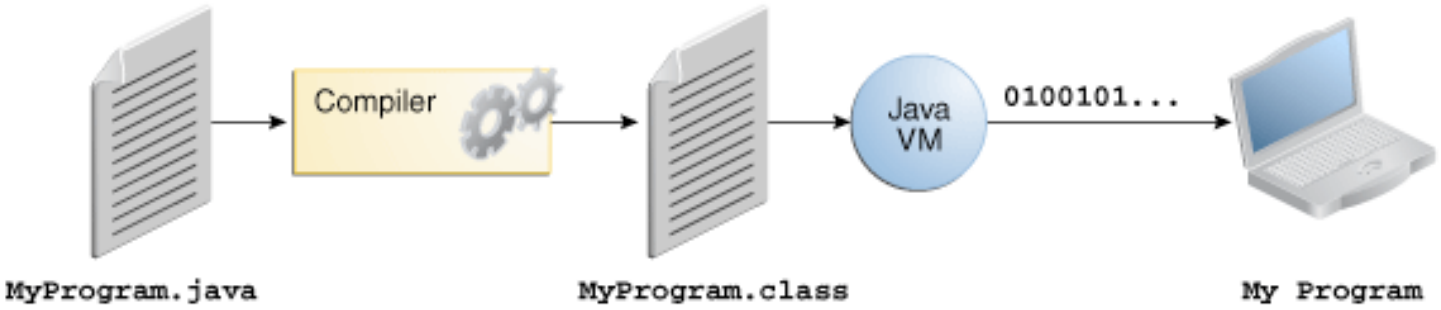
Software Development with Java

All source code is first written in plain text files ending with the ".java" extension.

Those source files are then compiled into ".class" files by the javac compiler.

A ".class" file does not contain code that is native to your processor; it instead contains *bytecodes* — the machine language of the Java Virtual Machine (Java VM).

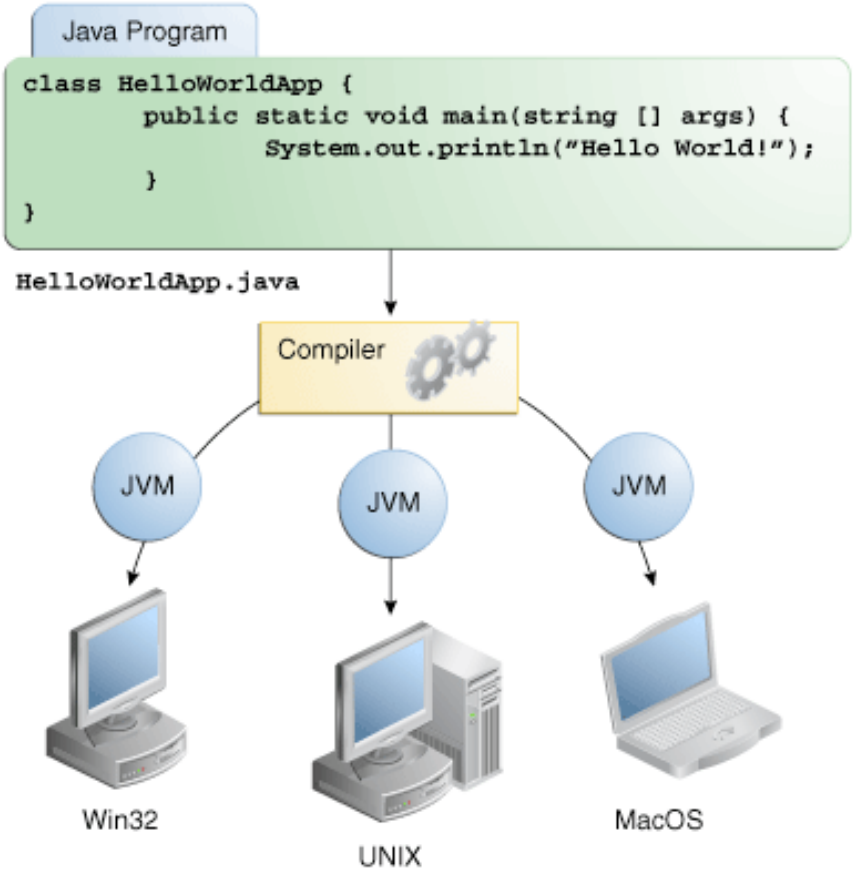
The java launcher tool then runs your application with an instance of the Java Virtual Machine, i.e. your code is run by JVM.



http://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html

Platform Independence: Write Once Run Anywhere

Because the Java VM is available on many different operating systems, the same .class files are capable of running on Microsoft Windows, the Solaris™ Operating System (Solaris OS), Linux, or Mac OS.



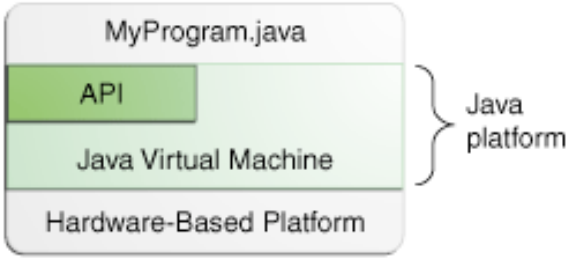
http://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html

The Java Platform

A *platform* is the hardware or software environment in which a program runs.

The Java platform has two components:

- The *Java Virtual Machine: I*t's the base for the Java platform and is ported onto various hardware-based platforms
- The *Java Application Programming Interface* (API): It is a large collection of ready-made software components that provide many useful capabilities.



As a platform-independent environment, the Java platform can be a bit slower than native code.

- However, advances in compiler and virtual machine technologies are bringing performance close to that of native code without threatening portability.

http://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html

Your First Java Program

HelloWorld.java

```
public class HelloWorld {

      public static void main(String[] args) {
            System.out.println("Hello world!");
      }

}
```

```
$ javac HelloWorld.java ← Compile
$ java HelloWorld       ← Run
Hello world!
```

Basic Programming Elements

Variables, Types and Expressions

Flow of Control

- Branching
- Loops

Variables

Variables in a program are used to store data such as numbers and letters. They can be thought of as containers of a sort.

You should choose variable names that are helpful. Every variable in a Java program must be declared before it is used for the first time.

A variable declaration consists of a type name, followed by a list of variable names separated by commas. The declaration ends with a semicolon.

```
Syntax:
data_type variable_name [ = initial_value ];
```

```
int styleNumber, numberOfChecks, numberOfDeposits;
double amount, interestRate;
char answer;
```

Primitive Data Types

Type Name	Kind of Value	Memory Used	Range of Values
byte	Integer	1 byte	-128 to 127
short	Integer	2 bytes	-32,768 to 32,767
int	Integer	4 bytes	-2,147,483,648 to 2,147,483,647
long	Integer	8 bytes	-9,223,372,036,8547,75,808 to 9,223,372,036,854,775,807
float	Floating-point	4 bytes	±3.40282347 × 10+38 to ±1.40239846 × 10–45
double	Floating-point	8 bytes	±1.79769313486231570 × 10+308 to ±4.94065645841246544 × 10–324
char	Single character (Unicode)	2 bytes	All Unicode values from 0 to 65,535
boolean		1 bit	True or false

There are also Class Data Types which we will cover later.

Identifiers

The technical term for a name in a programming language, such as the name of a variable, is an **identifier.**

An identifier can contain only letters, digits 0 through 9, and the underscore character "_".

The first character in an identifier cannot be a digit.

There is no limit to the length of an identifier.

Java is **case sensitive** (e.g., *personName* and *personname* are two different variables).

Identifier	Valid?
outputStream	Yes
4you	No
my.work	No
FirstName	Yes
_tmp	Yes
public	No

public is a reserved word.

Java Reserved Words

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

Naming Conventions

Class types begin with an uppercase letter (e.g. **String**).

Primitive types begin with a lowercase letter (e.g. **float**).

Variables of both class and primitive types begin with a lowercase letters (e.g. **firstName**, **classAverage**).

Multiword names are "punctuated" using uppercase letters.

Assignment Statements

An assignment statement is used to assign a value to a variable.

The "equal sign" is called the *assignment operator*

Syntax:

```
variable_name = expression;
```

where **expression** can be another variable, a *literal* or *constant*, or something to be evaluated by using *operators*.

```
amount = 100;
interestRate = 0.12;
answer = 'Y';
fullName = firstName + " " + lastName;
```

Initializing Variables

A variable that has been declared, but no yet given a value is said to be *uninitialized.*

Uninitialized class variables have the value **null**.

Uninitialized primitive variables may have a default value.

Data Type	Default Value
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
String (or any object)	null
boolean	FALSE

It's good practice **not** to rely on a default value.

Constants

Literal expressions such as **2**, **3.7**, or **'y'** are called *constants*.

Integer constants can be preceded by a **+** or **-** sign, but cannot contain commas.

Floating-point constants can be written with digits after a decimal point or using *e notation.*

- `765000000.0` can be written as `7.65e8`
- `0.000483` can be written as `4.83e-4`

Imprecision in Floating Point Numbers

Floating-point numbers often are only approximations since they are stored with a finite number of bits.

Hence **1.0/3.0** is slightly less than 1/3.

1.0/3.0 + 1.0/3.0 + 1.0/3.0 is less than 1.

Named Constants

Java provides a mechanism that allows you to define a variable, initialise it, and moreover fix the variable's value so that it cannot be changed.

```
public static final Type Variable = Constant;
```

The convention for naming constants is to use all uppercase letters, with an underscore symbol "_" between words.

```
public static final double PI = 3.14159;
public static final int DAYS_PER_WEEK = 7;
…
float area = PI * r * r ;
int daysInYear = 52 * DAYS_PER_WEEK ;
```

Assignment Compatibility

Java is *strongly typed.*

A value of one type can be assigned to a variable of any type further to the right (not to the left):

byte → short → int → long → float → double

You can assign a value of type **char** to a variable of type **int**.

Type Conversion (Casting)

Implicit conversion

```
double doubleVariable = 5;              // 5.0
int intVariable = 5;                    // 5
doubleVariable = intVariable;           // 5.0
```

Explicit conversion

```
double doubleVariable = 5.0;
int intVariable = doubleVariable ;              // Illegal
int intVariable = (int) doubleVariable ;        // Legal, 5
```

Operators and Precedence

Precedence

- First: The unary operators: plus (+), minus(-), not (!), increment (++) and decrement (--)
- Second: The binary arithmetic operators: multiplication (*), integer division (/) and modulus (%)
- Third: The binary arithmetic operators: addition (+) and subtraction (-)

When binary operators have equal precedence, the operator on the left acts before the operator(s) on the right.

When unary operators have equal precedence, the operator on the right acts before the operation(s) on the left.

Parenthesis can change the precedence.

Operators and Precedence - Example

Ordinary Math	Java (Preferred Form)	Java (Parenthesized)
$rate^2 + delta$	`rate * rate + delta`	`(rate * rate) + delta`
$2(salary + bonus)$	`2 * (salary + bonus)`	`2 * (salary + bonus)`
$\frac{1}{time + 3mass}$	`1 / (time + 3 * mass)`	`1 / (time + (3 * mass))`
$\frac{a-7}{t+9v}$	`(a - 7) / (t + 9 * v)`	`(a - 7) / (t + (9 * v))`

Figure from "Java - An Introduction to Problem Solving and Programming, Walter Savitch, Pearson, 2012"

Specialised Assignment Operators

You can precede the simple assignment operator (=) with an arithmetic operator (+, -, *, /, %) to produce a kind of special-purpose assignment operator.

```
amount += 5;      equals to      amount = amount + 5;
amount *= 5;      equals to      amount = amount * 5;
```

Increment / Decrement Operators

Used to increase (or decrease) the value of a variable by 1

The increment operator

- count++ → Use the value of count and then increase it.
- ++count → Increase the value of count and then use it.

The decrement operator

- count-- → Use the value of count and then decrease it.
- --count → Decrease the value of count and then use it.

Increment / Decrement Operators - Example

The increment operator (prefix form)

```
int n = 3;
int m = 4;
int result = n * (++m);  // result = 15
```

The increment operator (postfix form)

```
int n = 3;
int m = 4;
int result = n * (m++); // result = 12
```

Arrays

Array is a sequence of values.

Array indices begin at zero.

Defining Arrays

```
Base_Type[] Array_Name = new Base_Type[Length];
```

```
int[] numbers = new int[100];              // or,
```

```
int[] numbers;
numbers = new int[100];
```

Initializing Arrays

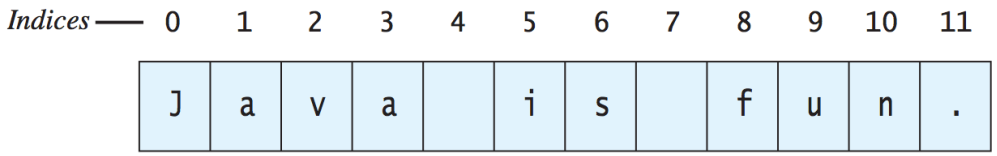
```
double[] reading = {3.3, 15.8, 9.7};     // or,
```

```
double[] reading = new double[3];
reading[0] = 3.3;
reading[1] = 15.8;
reading[2] = 9.7;
```

Strings

A value of type **String** is a

- Sequence (Array) of characters treated as a single item
- Character positions start with 0



Note that the blanks and the period count as characters in the string.

Can be declared in three ways:

```
String greeting;
greeting = "Hello World!";
```

```
String greeting = "Hello World!";
```

```
String greeting = new String("Hello World!");
```

Figure from "Java - An Introduction to Problem Solving and Programming, Walter Savitch, Pearson,

Concatenating Strings

You can connect—or join or paste—two strings together to obtain a larger string. This operation is called **concatenation** and is performed by using the "+" operator.

```
String greeting, sentence;
greeting = "Hello";

sentence = greeting + " my friend!";
System.out.println(sentence);     // Hello my friend!
```

```
String solution = "The answer is " + 42;
System.out.println(solution);     // The answer is 42

// Java converts the number constant 42 to the
// string constant "42" and then concatenates the
// two strings
```

String Methods

Homework: Investigate the methods given below. You will be responsible in the exams.

charAt (Index)	length()
compareTo(A_String)	replace(OldChar, NewChar)
concat(A_String)	substring(Start)
equals(Other_String)	substring(Start,End)
equalsIgnoreCase(Other_String)	toLowerCase()
indexOf(A_String)	toUpperCase()
lastIndexOf(A_String)	trim()

Boolean Type

Java has the logical type `boolean`

Type `boolean` has two literal constants

- `true`
- `false`

```
int number = -5;
boolean isPositive = (number > 0);        // False
```

Java Comparison Operators

Math Notation	Name	Java Notation	Java Examples
=	Equal to	==	`balance == 0` `answer == 'y'`
≠	Not equal to	!=	`income != tax` `answer != 'y'`
>	Greater than	>	`expenses > income`
≥	Greater than or equal to	>=	`points >= 60`
<	Less than	<	`pressure < max`
≤	Less than or equal to	<=	`expenses <= income`

Figure from "Java - An Introduction to Problem Solving and Programming, Walter Savitch, Pearson,

Java Logical Operators

Name	Java Notation	Java Examples
Logical *and*	&&	`(sum > min) && (sum < max)`
Logical *or*	||	`(answer == 'y') || (answer == 'Y')`
Logical *not*	!	`!(number < 0)`

Figure from "Java - An Introduction to Problem Solving and Programming, Walter Savitch, Pearson, 2012"

Flow of Control

Flow of control is the order in which a program performs actions.

A *branching statement* chooses between two or more possible actions.

- If-else, switch statements

A *loop statement* repeats an action until a stopping condition occurs.

- For, while, do-while loops

Basic if Statement

Syntax

```
if (Expression)
    Action
```

If the *Expression* is true then execute *Action*

Action is either a single statement or a group of statements within braces

```
if (value2 < value1) {      // Rearrange numbers so
      int tmp = value1;     // value2 variable should
      value1 = value2;      // hold the bigger value
      value2 = tmp;
}
```

if-else Statement

Syntax

```
if (Expression)
   Action1
else
   Action2
```

If *Expression* is true then execute *Action1* otherwise execute *Action2*

The actions are either a single statement or a list of statements within braces

```
int maximum;
if (value1 < value2) {    // is value2 larger?
   maximum = value2;      // yes: value2 is larger
}
else {                    // (value1 >= value2)
   maximum = value1;      // no: value2 is not larger
}
```

if-else-if Statement

If statements can be nested (also called as multi-way, multi-branch if statement)

```
if (a == '0')
       System.out.println ("zero");
else if (a == '1')
       System.out.println ("one");
else if (a == '2')
       System.out.println ("two");
else if (a == '3')
       System.out.println ("three");
else if (a == '4')
       System.out.println ("four");
else
       System.out.println ("five+");
```

Switch Statement

Switch statement can be used instead of multi-way if statement.

Syntax

```
switch(controlling_expression) {
        case expression1:
                action1;
                break;
        case expression2:
                action2;
                break;
        …
        default:
                actionN;
}
```

Every case ends with ***break*** statement.

Switch Statement

Switch statements are more readable than nested if statements

```
switch (a) {
    case '0':
        System.out.println ("zero"); break;
    case '1':
        System.out.println ("one"); break;
    case '2':
        System.out.println ("two"); break;
    case '3':
        System.out.println ("three"); break;
    case '4':
        System.out.println ("four"); break;
    default:
        System.out.println ("five+"); break;
}
```

The Conditional (Ternary) Operator

The `?` and `:` together are called the *conditional operator* or *ternary operator.*

```
if (n1 > n2)
   max = n1;
else
   max = n2;
```

can be written as:

```
max = (n1 > n2) ? n1 : n2;
```

for Loops

The for loop is a pretest loop statement. It has the following form.

```
for (initialisation; boolean-expression; increment){
    nested-statements
}
```

initialisation is evaluated first.

boolean-expression is tested *before* each iteration of the loop.

increment is evaluated at the end of each iteration.

nested-statements is a sequence of statements. If there is only one statement then the braces may be omitted

Varying Control Variable

for (int i = 1; i <= 100; i++)

- from 1 to 100 in increments of 1

for (int i = 100; i >= 1; i--)

- from 100 to 1 in increments of -1

for (int i = 7; i <= 77; i += 7)

- from 7 to 77 in increments of 7

for (int i = 20; i >= 2; i -= 2)

- from 20 to 2 in decrements of 2

For Loop Example

```
String[] classList = {"Jean", "Claude", "Van",
                      "Damme"};

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

```
Jean
Claude
Van
Damme
```

```
for (String name : classList) {
      System.out.println(name);
}
```

```
Jean
Claude
Van
Damme
```

While Loop

The while loop is a pretest loop statement. It has the following form.

```
while (boolean-expression) {
        nested-statements
}
```

boolean-expression is an expression that can be true or false.

nested-statements is a sequence of statements. If there is only one statement then the braces can be omitted.

The boolean expression is tested *before* each iteration of the loop. The loop terminates when it is false.

While Loop Example

```
int[] numbers = { 1, 5, 3, 4, 2 };
int i=0, key = 33;

boolean found = false;

while (!found){
      if (numbers[i++] == key)
            found=true;
}

if (found)
      System.out.println("Key is found in the array");
else
      System.out.println("Key is NOT found!");
```

Let's look for something that does not exist.

Is there a problem here?

While Loop Example

```
int[] numbers = { 1, 5, 3, 4, 2 };
int i=0, key = 33;

boolean found = false;

while (!found && i<numbers.length){
      if (numbers[i++] == key)
            found=true;
}

if (found)
      System.out.println("Key is found in the array");
else
      System.out.println("Key is NOT found!");
```

Make sure that the loop ends somehow.

Do-While Loop

The do-while loop is a post-test loop statement. It has the following form.

```
do {
        nested-statements
} while (boolean-expression);
```

nested-statements is a sequence of statements. If there is only one statement then the braces may be omitted.

boolean-expression is an expression that can be true or false.

The boolean expression is tested *after* each iteration of the loop. The loop terminates when it is false.

Do-While Example

```
Scanner scan = new Scanner(System.in);
int myNumber;

do {
      System.out.println(
            "Enter a number between 0 and 100: ");

      myNumber = scan.nextInt();

} while (!(myNumber >= 0 && myNumber <= 100));

System.out.println("You entered a valid number");
```

Break Statement

The break statement is used in loop (for, while, and do-while) statements and switch statements to terminate execution of the statement. A break statement has the following form.

```
break;
```

After a break statement is executed, execution proceeds to the statement that follows the enclosing loop or switch statement.

Use **`break`** statements sparingly (if ever).

Continue Statement

A **continue** statement

- Ends current loop iteration
- Begins the next one

Use of continue statement is not recommended

- Introduce unneeded complications

Breaking a Loop

```
int[] numbers = { 1, 5, 3, 4, 2 };
int i = 0, key = 3;

while (i < numbers.length) {
      if (numbers[i] == key)
            break;
      i++;
}

if (i < numbers.length)
      System.out.println("Key is found in the array");
else
      System.out.println("Key is NOT!");
```

Summary

So far, it should be fairly easy to follow for those who has basic programming skills / who has taken BBM101.

We will continue with objects next week.

In the mean time, here is a good starting point to Java:

http://docs.oracle.com/javase/tutorial/index.html

Also check out these notes by Oğuz Aslantürk in Turkish:

http://web.cs.hacettepe.edu.tr/~bbm102/misc/java_notes_by_oa.pdf

Acknowledgments

The course material used to prepare this presentation is mostly taken/adopted from the list below:

- Java - An Introduction to Problem Solving and Programming, Walter Savitch, Pearson, 2012.
- Java tutorials http://docs.oracle.com/javase/tutorial/
- Aaron Bloomfield, CS101, University of Virginia.