PROGRAMMING III

JAVA LANGUAGE

COURSE 1

COURSE CONTENT

- OOP Concepts. Java Language
- Classes
- Comparing objects in Java
- Collections. Generics
- Graphical Interfaces. Swing
- Java IO
- JDBC - Java Database Connectivity
- Threads

ORGANIZE STUFFS

- **Course**
    - Flavia Micota
        - cab. 046B
        - contact: flavia.micota@e-uvt.ro
        - Site: http://staff.fmi.uvt.ro/~flavia.micota
  - Laboratory
    - Flavia Micota
    - Valentin Pop
- **Consultation timetable**
    - Monday 8:30-9:30 050A
    - Thursday 9:30-10:30 032
- **Attendee**
    - Course
        - random tests from subjects presented in current course
    - Laboratory
        - Minimum 7 presences
    - Recontracting
        - IF (number of presences at laboratory < 4) THEN recontract

ORGANIZE STUFF

- **Mark**
  - Theoretical exam 50%
  - Laboratory test after exam 30%
  - Homework - 10%
  - Attendee - 10%
    - 5% course tests
    - 5% laboratory activity/test

- **Homework**
  - submit: classroom code - **p83ar7j**
  - cut date: 2 weeks from the moment of announcement

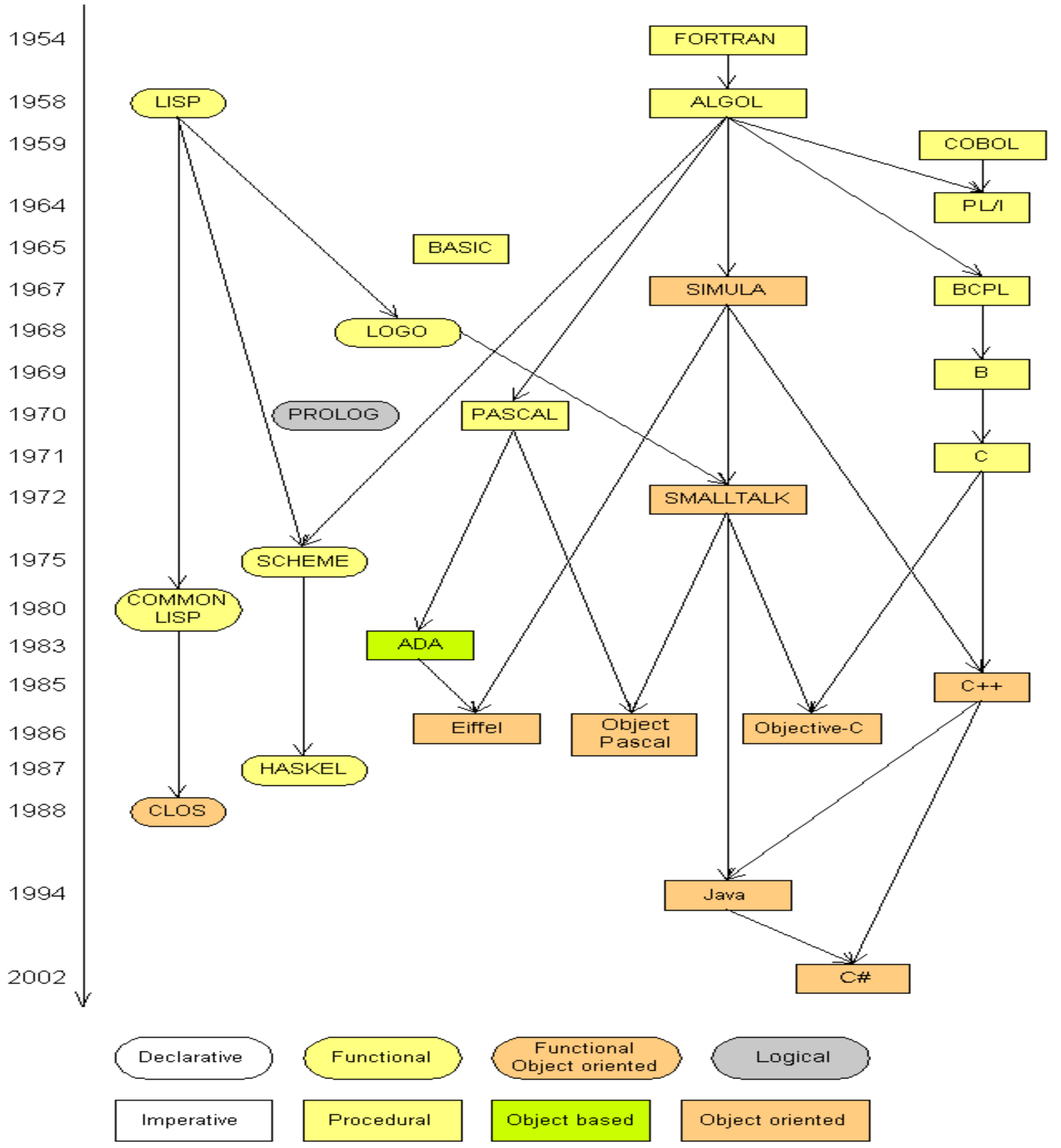
COURSE 1. CONTENT

- ❑ Object Oriented Programming
- ❑ Java Language History
- ❑ Java Program Structure
- ❑ Java Language

PROGRAMMING LANGUAGES

- **Imperative (algorithmic) languages**
  - The program is a sequence of statements
  - Uses variables to access memory
  - Types
    - Procedural Languages
    - Object Oriented Languages
- **Declarative (non-algorithmic) languages**
  - The programmer presents the problem, the way to solution it is included in the language
  - Types
    - Functional (applicative) languages
    - Logic languages
- **Other languages**

PROGRAMMING LANGUAGES



PROGRAMMING PARADICSM

- ❑ Unstructured programming
- ❑ Procedural programming
- ❑ Modular programming
- ❑ Data abstractization
- ❑ Object oriented programming
- ❑ Generic programming (templates)
- ❑ Aspected oriented programing (AOP)

OBJECT ORIENTED LANGUAGE

A language or technique is object-oriented if and only if it directly supports

[Stroustrup, 1995]:

[1] Abstraction – providing some form of classes and objects

[2] Inheritance – providing the ability to build new abstractions out of existing ones

[3] Runtime polymorphism – providing some form of runtime binding.

OBJECT ORIENTED LANGUAGE

- **Objects**
  - Have a state that reflects by current characteristics and conditions and a behaviour that describe the action that it cat execute
- **Classes**
  - Groups objects with similar characteristics
- **Data Encapsulation**
  - Hidding object data and behaviour
- **Data Abstractization**
  - A simplification or a model of a compex concept, process or real word object
- **Inheritance**
  - Is a contract between a class and the outside world
  - When a class implements an interface, it promises to provide the behavior published by that interface
- **Polymorphism**
  - The possibility to offer an interface that has different implementations for different objects

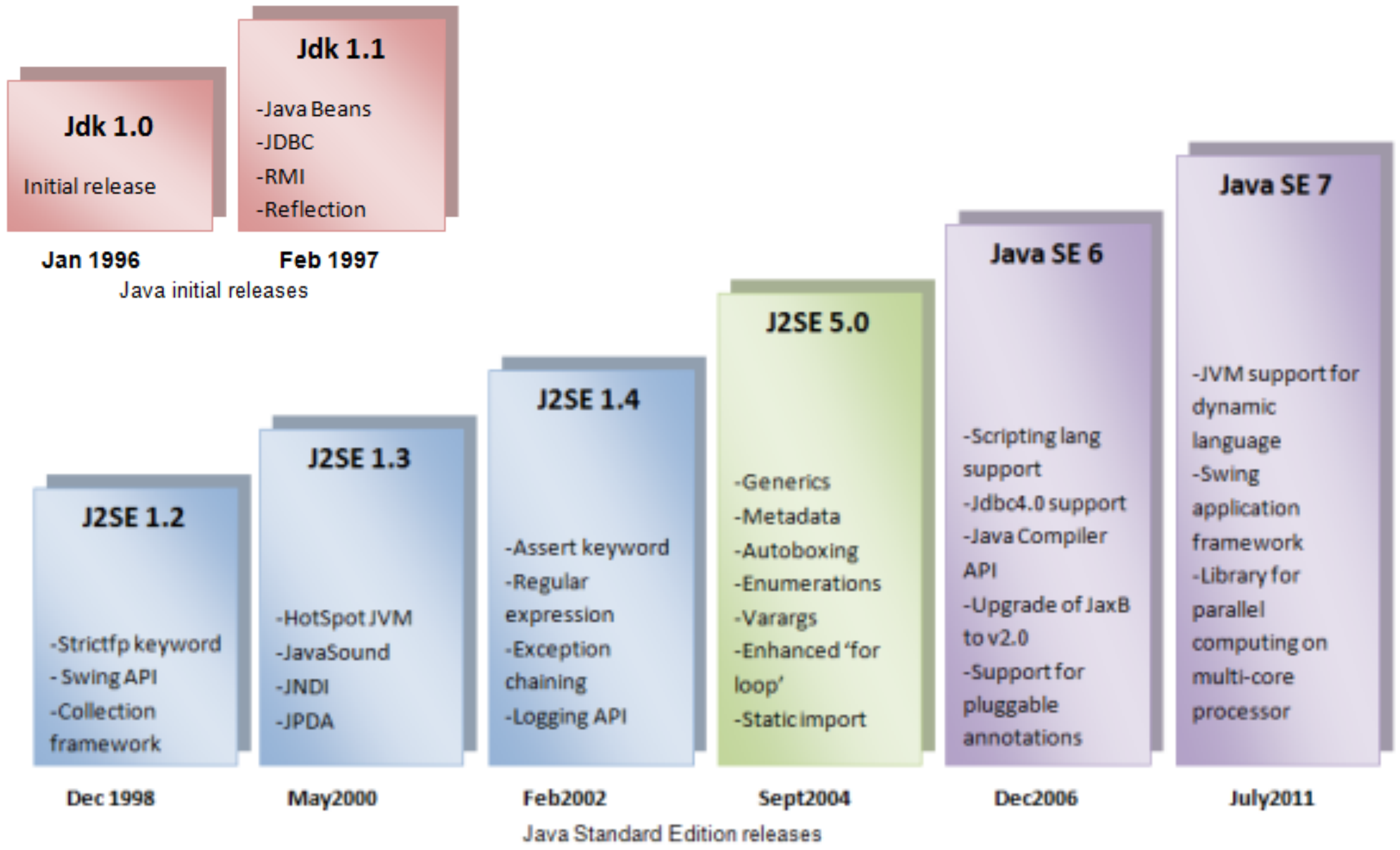
OBJECT ORIENTED LANGUAGE

- **Objects**
  - Have a state that reflects by current characteristics and conditions and a behavior that describe the action that it can execute
- **Classes**
  - Groups objects with similar characteristics
- **Data Encapsulation**
  - Hiding object data and behavior
- **Data Abstraction**
  - A simplification or a model of a complex concept, process or real word object
- **Inheritance**
  - Is a contract between a class and the outside world
  - When a class implements an interface, it promises to provide the behavior published by that interface
- **Polymorphism**
  - The possibility to offer an interface that has different implementations for different objects

JAVA PLATFORMS

- **J2SE (Standard Edition)**
  - offers support for creating desktop applications and applets
  - Contains the standard set of classes offered by Java
- **J2ME (Micro Edition)**
  - offers support for programming on mobile devices
- **J2EE (Enterprise edition)**
  - Offers support for complex applications on web. It contains standards for database accessing, servlets, beans, web services, messages queues ...
- **Site**
  - http://www.oracle.com/technetwork/java/index.html

JAVA LANGUAGE EVOLUTION



Jdk 1.0

Initial release

Jan 1996

Jdk 1.1

-Java Beans
-JDBC
-RMI
-Reflection

Feb 1997

Java initial releases

J2SE 1.2

-Strictfp keyword
- Swing API
-Collection framework

Dec 1998

J2SE 1.3

-HotSpot JVM
-JavaSound
-JNDI
-JPDA

May2000

J2SE 1.4

-Assert keyword
-Regular expression
-Exception chaining
-Logging API

Feb2002

J2SE 5.0

-Generics
-Metadata
-Autoboxing
-Enumerations
-Varargs
-Enhanced 'for loop'
-Static import

Sept2004

Java SE 6

-Scripting lang support
-Jdbc4.0 support
-Java Compiler API
-Upgrade of JaxB to v2.0
-Support for pluggable annotations

Dec2006

Java SE 7

-JVM support for dynamic language
-Swing application framework
-Library for parallel computing on multi-core processor

July2011

Java Standard Edition releases

IDE JAVA

- **NetBeans**
- **Eclipse**
  - https://eclipse.org/
- **IntelliJ**
- **BlueJ**
  - developed mainly for educational porpuse

JAVA APPLICATIONS

- **Stand alone**
  - Contains main() method
  - Compile
    - `javac fileName.java`
  - Execution
    - `java fileName`
- **Applets**
  - Inherits `Applet` or `JApplet` class
  - Compile
    - `javac fileName.java`
  - Execution
    - create a HTML page that contains tag APPLET that refers to compiled class
    - `appletviewer html.page`
    - Java Web Start
- **Servlets**
  - Inherates class `HttpServlet`
  - Compile
    - `javac fileName.java`
  - Execution
    - an WAR archive deployed on a WEB Server
  - NOT object of this course

JAVA PROGRAM STRUCTURE

```
[package identifier;]


[import class;]


[access specifiers] class/interface ClassName {
    //member attributes declaration
   // member methods declaration
}
```

All code (functions, variable declarations) is included inside a java class. It can't exist code outside a class.

If a class is declared to be public it must be placed in a file with same name like the class

FIRST EXAMPLE

Starting point of a desktop application in Java.
The signature of the method cannot be changed

File: Example.java

```
public class Example {
        public satic void main (String args[]) {

                System.out.println ( “Hello World!”);
        }
}
```

The method **println**() that belong to class **out** displays a text to standard output

Compile

```
        javac Example.java => Example.class
```

Execution

```
        java Example
```

Output

```
        Hello World!
```

JAVA CODDING GUIDELINES

- **Different standards**
  - http://www.oracle.com/technetwork/java/codeconventions-135099.html
  - https://google.github.io/styleguide/javaguide.html
  - https://www.securecoding.cert.org/confluence/display/java/Java+Coding+Guidelines

JAVA CODDING GUIDELINES

- **Packages**
  - the prefix of a unique package name is always written in all-lowercase
- **Classes**
  - should be nouns
  - in mixed case with the first letter of each internal word capitalized
- **Interfaces**
  - names should be capitalized like class names
- **Methods**
  - should be verbs
  - in mixed case with the first letter lowercase, with the first letter of each internal word capitalized
- **Variables**
  - should not start with '_'
  - the name starts with lower case
  - each word starts with upper case
- **Constants**
  - should be uppercase with words separated by underscores ('_')

JAVA KEYWORDS

Category	Keyword	Example
Primitive Types	boolean	`boolean isopen = true;`
	byte	`byte i1 = -128;`
	char	`char c ='A';`
	short	`short i =10;`
	int	`int i = 10;`

	long	`long i  = 7l;`
		`long j = 1234567567;`

	float	`float i =3.4f;`
	double	`double i = 3.4;`

JAVA KEYWORDS

Category	Keyword	Example

Control Flow	for	for(int i=0; i<10; i++){
		...}

	do while	do{ ... }while (i<10);
	while	while (true) { ... }

	if	if (a<3) { ...
		} else if (a>5) { ...
		} else { ... }

	else	

	switch	swich(i) {
		    case "abc": ...
		        breack;
		    default:
		        ...
		}

	case	
	default	

JAVA KEYWORDS

Category	Keyword	Example
Control flow	break	`break label;`
	continue	`continue label;`
	return	`return i;`
	try	`try{` `...`
	throw	`throw new Exception();` `...`
	catch	`} catch (Exception e) {` `...`
	finally	`} finally {` `...` `}`
	throws	`void fct () throws Exception { ... }`

JAVA KEYWORDS

Category	Keyword	Example
Modifier	public	`public int i;`
	protected	`protected int i;`
	private	`private int i;`
	static	`static int i;`
	final	`final int i;`
	abstract	`abstract void fct() {  ... }`

	synchronized	`synchronized int funct() { ... }`
		`synchronized (obj) { .. }`

	native	`native int funct() { ... }`
	tansient	`transient int i;`
	volatile	`volatile int i;`

JAVA KEYWORDS

Category	Keyword	Example
Classes	class	`class A { ... }`
	interface	`interface A { ... }`
	extends	`class A extends B { ... }`
	implements	`class A implements B { ... }`
	package	`package ro.uvt.p3;`
	import	`import java.awt.*;`

OBS: Some of the modifiers keywords can be used together with classes not just with class fields.

JAVA KEYWORDS

Category	Keyword	Example
Miscellaneous	(true)	`boolean x = true;`
	(false)	`boolean x = false;`
	(null)	`Object obj = null;`
	void	`void fct( ) { ... }`
	this	`this.x = x;`
	new	`Object obj = new Object();`
	super	`super ("call base classs constructor")`
	instanceof	`if (a instanceof String)` `String s = (String) a;`

OPERATORS

Category	Operator	Description
Simple Assigment	=	Simple assigment operator
Aritmetic	+	Additive (also used for String concatenation)
	-	Substraction
	*	Multiplication
	/	Division
	%	Remainder
Unary	+	Indicates positive value
	-	Negates a value
	++	Increment
	--	Decrement
	!	Logical complement

OPERATORS

Category	Operator	Description
Equality and Relational	==	Equal to
	!=	Not equal to
	>	Greater then
	>=	Greater then or equal to
	<	Less then
	<=	Less then or equal to
Conditional	&&	Conditional AND
	||	Conditional OR
	?:	Ternary (if - then - else)

OPERATORS

Category	Operator	Description
Type comparation	instanceof	Simple assigment operator
Bitwise and Bit Shift	~	Unary bitwise complement
	<<	Signed left shift
	>>	Signed right shift
	>>>	Unsigned right shift
	&	Bitwise AND
	^	Bitwise exclusive OR
	|	Bitwise inclusive OR

COMMENTS

- **Line comment**
  - //
- **Block comment**
  - /* */
- **Java Doc**
  - class documentation
  - methotds documentation

JAVADOC. CLASS COMMENTS

```
/**
* <h1>Add Two Numbers!</h1>
* The AddNum program implements an application that
* simply adds two given integer numbers and Prints
* the output on the screen.
* <p>
* <b>Note:</b> Giving proper comments in your program makes it more
*  user friendly and it is assumed as a high quality code. *
*  @author Popescu Ion
*  @version 1.0
*  @since 2016-08-31 */

public class AddNum { ...
}
```

JAVADOC

- ❑ Method comments
- ❑ Fields comments

```
/**
*  This method is used to add two integers. This is
*  a the simplest form of a class method, just to
*  show the usage of various javadoc Tags.
*  @param numA This is the first paramter to addNum method
*  @param numB This is the second parameter to addNum
method
*  @return int This returns sum of numA and numB. */
public int addNum(int numA, int numB) { ... }


/**
* This is the main method which makes use of addNum method.
* @param args Unused.
* @return Nothing.
* @exception IOException On input error.
* @see IOException
*/
public static void main(String args[]) throws
IOException { ... }
```

JAVAD0C. ANNOTATIONS

@author
@deprecated
@exception
@param

@return

@see

@since

@throws

@version

...

JAVADOC. GENERATING DOCUMENTATION

- **javadoc**
  - tool that allows generation of HTML pages based on javadoc annotations

  - Example
    - run in commned line: `javadoc AddNum.java`
    - result: a structure similar with official Java API documentation

JAVA UTIL STUFFS

- String class
- Display information on standard output \
- Autoboxing
- Math class
- Random numbers generation

STRING CLASS

- **java.lang.String**
  - stores charctes arrays
  - inmutable objects
    - the objects of the class cannot be modified
    - see: https://docs.oracle.com/javase/tutorial/essential/concurrency/ imstrat.html
- **Exemple**
  - `String s1 = null; //decleare a null string object`
  - `String s2 = “Course Java”; //declares and initialize a string object`

IMMUTABLE PATTERN

- **Don't provide "setter" methods — methods that modify fields or objects referred to by fields.**
- **Make all fields final and private.**
- **Don't allow subclasses to override methods.**
  - The simplest way to do this is to declare the class as final.
  - A more sophisticated approach is to make the constructor private and construct instances in factory methods.
- **If the instance fields include references to mutable objects, don't allow those objects to be changed**
  - Don't provide methods that modify the mutable objects.
  - Don't share references to the mutable objects. Never store references to external, mutable objects passed to the constructor; if necessary, create copies, and store references to the copies. Similarly, create copies of your internal mutable objects when necessary to avoid returning the originals in your method

STRING CLASS

- **Methods**
  - concatenation: “+”
    - `String s = “Course” + ' ' + “Java.”`
  - transformatios: `toUpperCase(), toLowerCase()`
    - `s.toLowerCase()`
  - comparations: `compareTo(), equals(), equalsIgnoreCase()`
    - `s.equalsIgnoreCase(“course java.”)`
  - search a string into a string: `contains(), endsWith(), indexOf(), lastIndexOf()`
  - operations: `split(), replace(), substring()`
  - size: `length()`

DISPLAY TO STANDARD OUTPUT

- **non-formated**
  - `System.out.print()`
    - `System.out.print(“without new line at the end”);`
  - `System.out.println()`
    - `System.out.print(“with new line at the end”);`
- **formated**
  - `System.out.printf([format], [value list])`
    - `System.out.printf("Integer : %d\n",15);`
    - `System.out.printf("String: %s, integer: %d, float: %.6f", "Hello World",89,9.231435);`
    - `System.out.printf("%-12s%-12s%s\n","Column 1","Column 2","Column3");`
  - OBS: String can be formatted to be used latter
    - `String s = String.format("%-12.5f%.20f", 12.23429837482,9.10212023134);`

AUTOBOXING

- **Concept related to generics (templates in C)**
- **For each basic type there is a corresponding class**

Basic Type	Corresponding Class
char	Characer
int	Integer
float	Float
double	Double
boolean	Boolean
byte	Byte
long	Long
short	Short

AUTOBOXING

- **before autoboxing**

```
Integer iObject = Integer.valueOf(3);

int iPrimitive = iObject.intValue() ;
```

- **after Java5**

```
Integer iObject = 3; //autobxing - primitive to wrapper conversion

int iPrimitive = iObject; //unboxing - object to primitive conversion
```

Each class that coresponds to a primitive type contains static methods to transform String objects to primitive types. ie.

```
int i = Integer.parseInt("123");
```

MATHEMATIC OPERATIONS

- **java.util.Math**
- **Static methods and constants**
  - Math.sqrt()
  - Math.abs()
  - Math.cons()
  - Math.random()
    - generates random numbers in [0,1)
  - ...
  - Math.PI
  - Math.E

RANDOM NUMBERS GENERATION

- **Using Math class**
  - `Math.random()`
    - generates uniform distributed numbers in [0,1)
- **Using Random class**
  - Pakage: `java.util.Random`
  - In order to user Random class create an object of type Random and call methods to generate random numbers
    - `Random r = new Random();`
  - Random class methods
    - `setSeed(long seed);`
    - `nextInt()`
      - generates uniform distributed numbers in [0, +2 147 483 647) (for 32 bytes)
    - `nextInt(value)`
      - generates uniform distributed numbers in [0, value)
    - `nextDouble()`
      - generates uniform distributed numbers in numbers in [0,1)
    - `nextBoolean()`

NEXT COURSE

- **Classes**
- **Objects**
- **Object class**

- **Access control specifiers**
  - fields
  - methods
  - classes