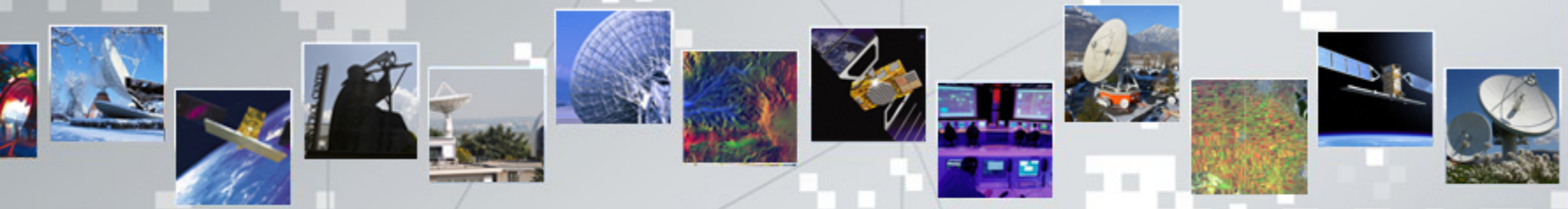


Applications of Computational Techniques for Physics

Java for Physics

Tecniche di calcolo per la Fisica

Modulo Java

Dr. Jacopo Pellegrino

TELESPAZIO
a LEONARDO and THALES company

A.A. 2017-2018

UNIVERSITÀ
DEGLI STUDI
DI TORINO

Contacts

- Dr. Jacopo Pellegrino
  - jacopo.pellegrino@telespazio-vega.de
  - jacopo.pellegrino@unito.it
  - Skype: jacopo.pellegrino13
  - http://personalpages.to.infn.it/~japelleg/

Always put “**TCF2018**” in the subject!!!

- Students help: right after lessons or anytime via mail or Skype

TELESPAZIO
a LEONARDO and THALES company

Who are we?

And what do we do?

Do we tend crops?

TELESPAZIO
a LEONARDO and THALES company

Do we forecast that sunny day at the beach?

TELESPAZIO
a LEONARDO and THALES company

Do we guide you through the skies?

TELESPAZIO
a LEONARDO and THALES company

Do we entertain you?

TELESPAZIO
a LEONARDO and THALES company

We are inspired by our visions and dreams.

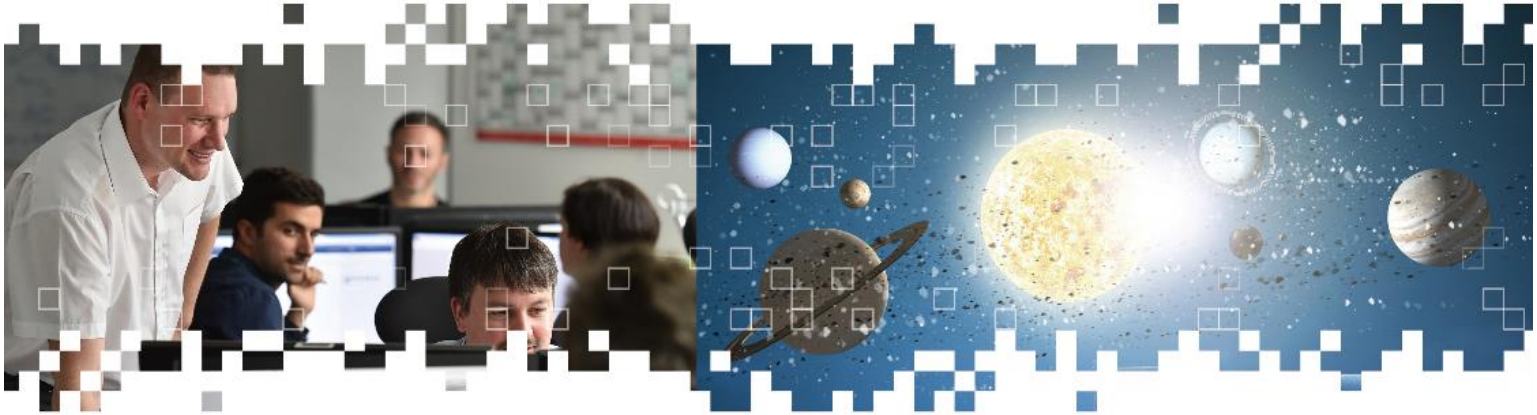
TELESPAZIO
a LEONARDO and THALES company

So, what do we do…

at Telespazio VEGA?

We are Telespazio VEGA

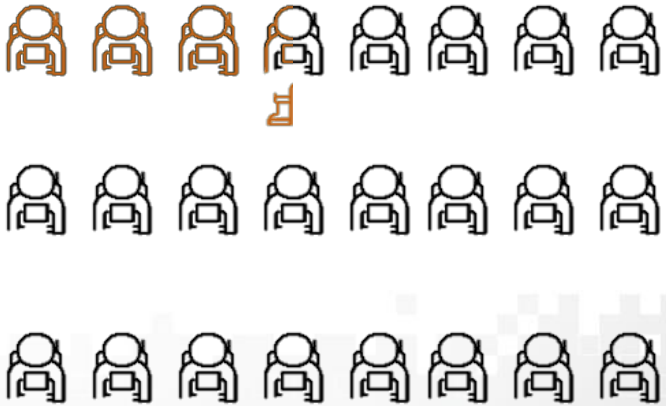
Our company



TELESPAZIO
a LEONARDO and THALES company

We are part of the Telespazio Group.

Our people



The Telespazio Group has 2,500 employees worldwide.

Including 350 employees in Telespazio VEGA Deutschland.

Turning visions into reality

Hollywood

Science-Fiction becomes reality

In 2016, the Rosetta mission comes to an end. After landing a probe, Philae, ESA plans to soft land the Rosetta orbiter on the comet Churyumov-Gerasimenko/67, ending the 12 year mission.

The science data will provide scientists with information about the chemical composition of the comet, especially looking at elements that are essential for life on Earth.

For the Rosetta and Philae mission we were involved in the development of the mission control system, planning system, the operational simulator, simulation-based training, flight control and flight dynamics.

Turning visions into reality

Weather

Even better forecasts

The Sentinels, a new fleet of ESA satellites, are delivering a wealth of data and imagery.

They are central to Europe's Copernicus programme to provide an all-weather, day-and-night supply of images of the Earth's surface.

We contribute to this project in the areas of ground segment systems, systems engineering and ICT. We are in charge of the routine operations.

Turning visions into reality

Simulated helicopter training

Maintenance Technicians must be trained

Telespazio VEGA Deutschland is developing the Aircraft Systems Trainer (AST) for the MRH90 transport helicopter.

It will be used by the Rotary Wing Aircraft Maintenance School (RAMS) of the Australian Army in Oakey, Queensland, Australia.

Within the NH90 training product line, we have previously developed a Virtual Maintenance Trainer (VMT).

Turning visions into reality

Internet from space

Going online anywhere – anytime

Terrestrial solutions tend to have only a limited capacity and reach. They are constantly exposed to harsh weather conditions or even natural disasters.

A space-based system can offer the best stability, the widest applicability and an easy usability.

Especially for B2B clients we are offering highly sophisticated solutions to guarantee their internet connection no matter where and when, with our partner Vodafone.

Turning visions into reality

agriloc

Automatic driving

No more than 100 years ago, our society and economy had its focus on farming crops which was hard and time consuming work.

In the future, tractors will drive and work autonomously while monitoring crops and their conditions.

We provide a reliable satellite connection between tractor and online agricultural services, to ensure accurate positioning.

TELESPAZIO
a LEONARDO and THALES company

Turning visions into reality

Enabling the NewSpace economy

ENABLE

We empower space ventures.

Advanced systems. Smart operations. As a service.

Disrupting the market of space operations

In the current NewSpace era, a revolution is required to drastically improve the cost and performance of space operations to launch and mature their business.

We are building the **world's first** cloud-based end-to-end platform for space operations, which will allow Mission-as-a-Service offerings to NewSpace organisations

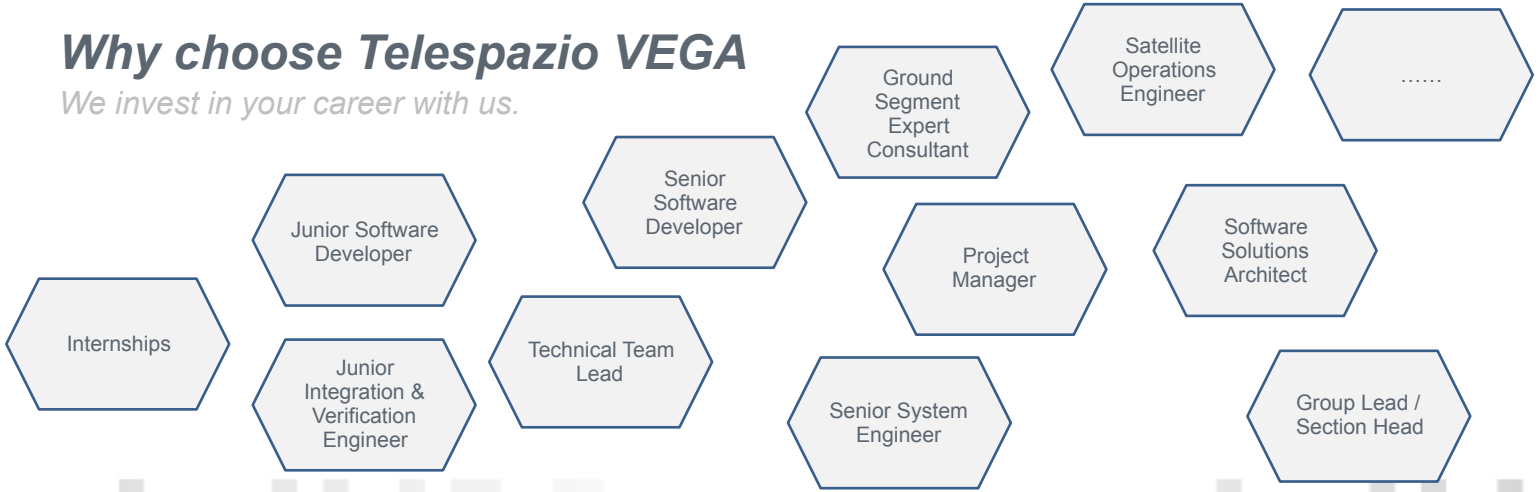
Be a part of our World!

Job profiles and Company culture

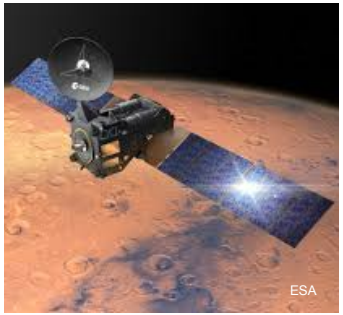
TELESPAZIO
a LEONARDO and THALES company

Why choose Telespazio VEGA

We invest in your career with us.

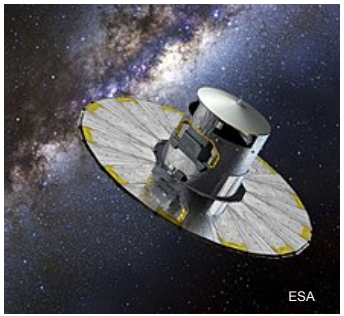


TELESPAZIO
a LEONARDO and THALES company



Interplanetary Missions

ExoMars TGO MCS



Astronomy Missions

Gaia MCS

Air Traffic Control

Integration of all ATC system components

Mission Control Systems for Space and Aviation

You are a graduate or going to graduate?

We are looking for you!

Software Solutions		
Computer Engineering, Information Technology, Natural Sciences or similar	Desirable: Good command of C++ or Java	Enthusiasm and innovative ideas

ICT Solutions & Services			
Systems Engineers	Network Engineers	IT Security/ Cybersecurity Engineers	Junior Engineers to Solution Architects (Project Managers)

Why choose Telespazio VEGA

Be(come) a part of our World!

Living in Germany

Close to Frankfurt/Main and Munich

Located in the heart of Europe

Populous and financially powerful country within the EU

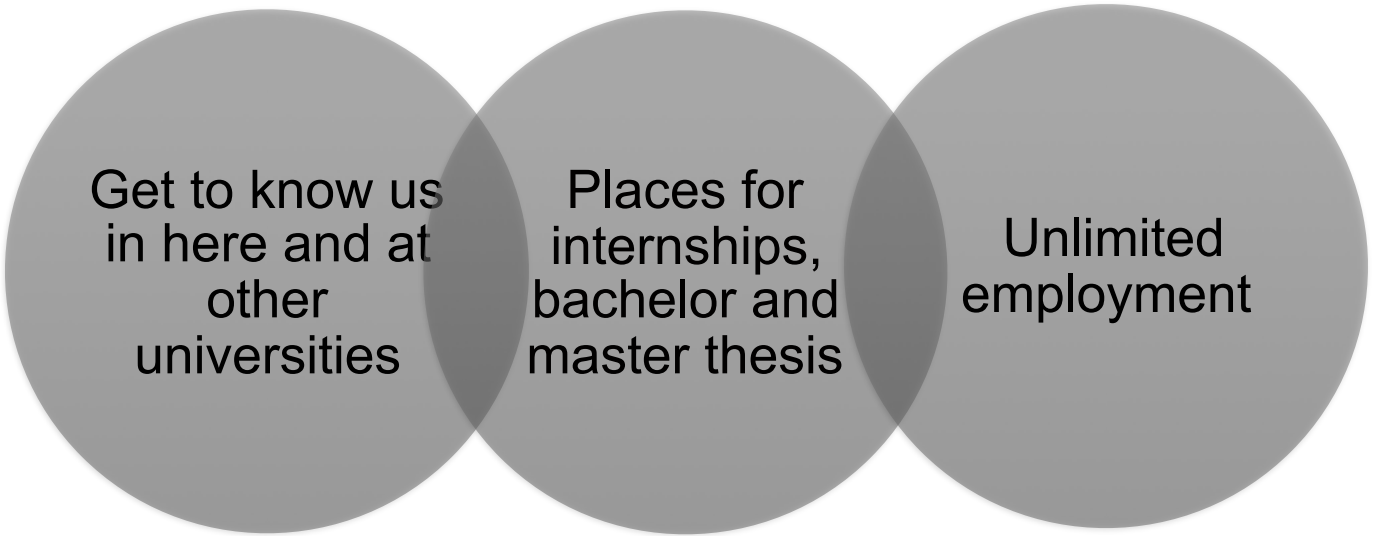
Temperate seasonal climate

Vast variety of landscapes, history, and attractions

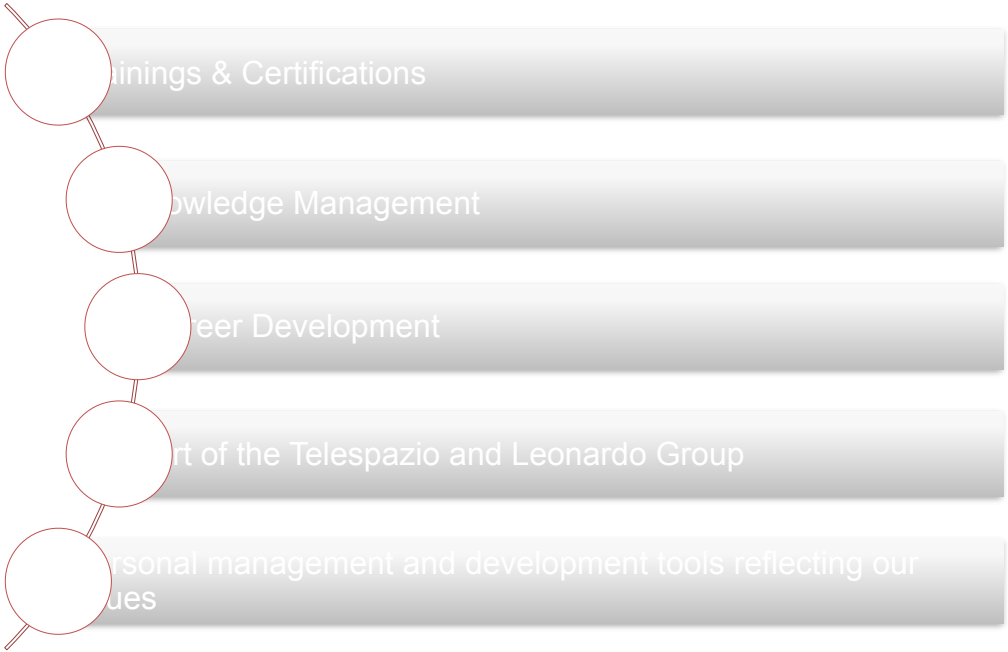
Rich tapestry of architecture, cuisine, folklore and traditions

What we offer to YOU! The key facts

Start working with us



Grow with us


inings & Certifications
owledge Management
eer Development
rt of the Telespazio and Leonardo Group
rsonal management and development tools reflecting our
ues

Every day, we face new challenges and as a result, we evolve.

www.telespazio-vega.de/en

TELESPAZIO
a LEONARDO and THALES company

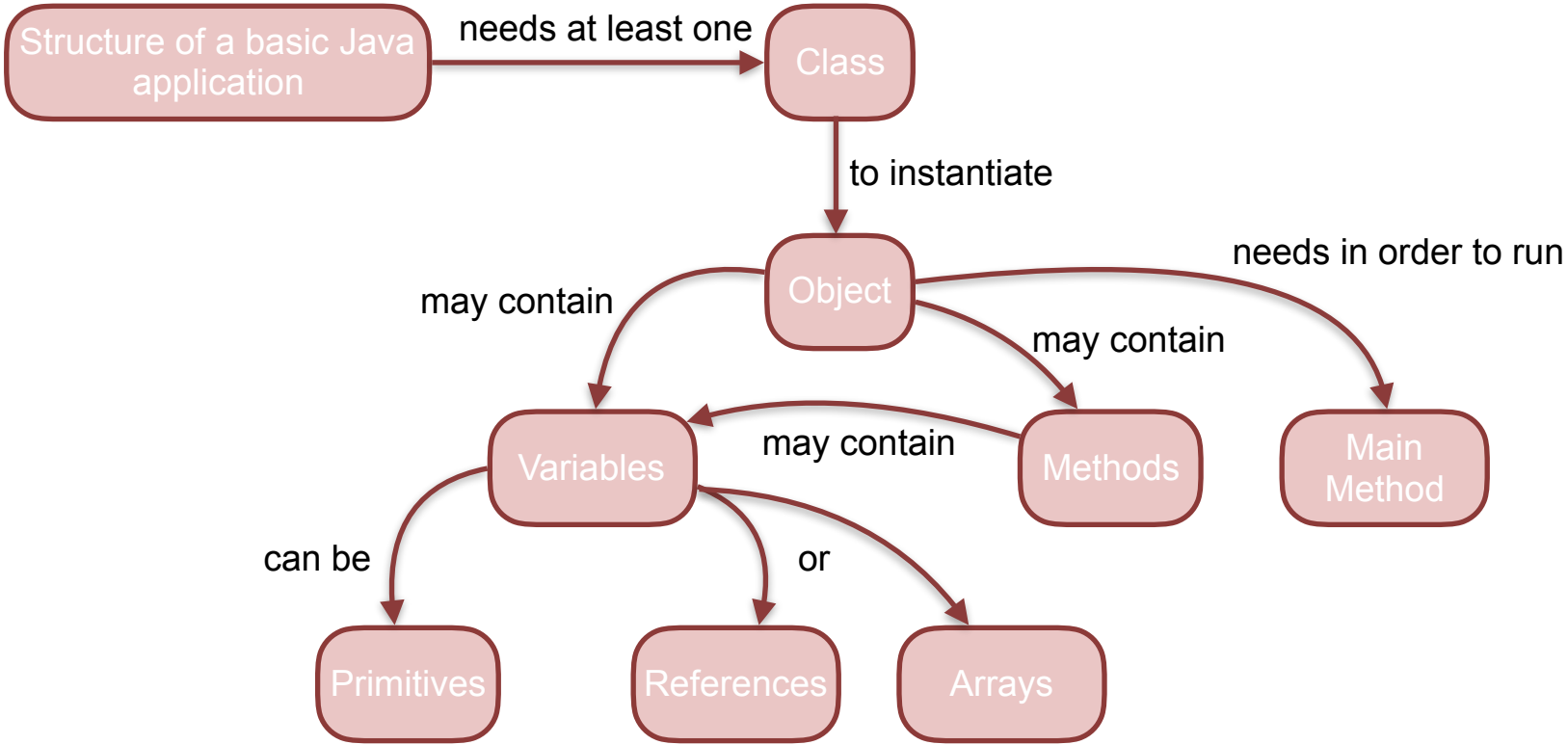
TELESPAZIO
a LEONARDO and THALES company

Introduction

Basic Java components

The building blocks of a Java application

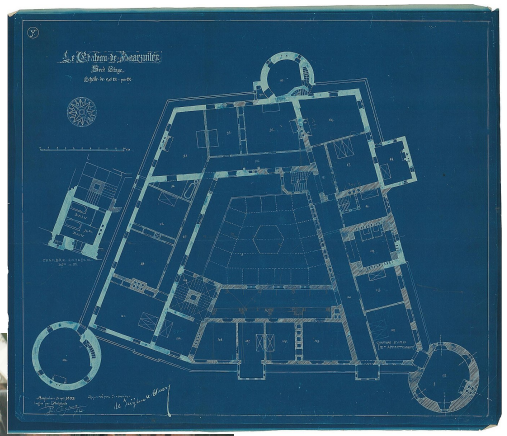
Structure of a basic Java application



Class and Object

A **class** is the definition or model of an object:

- it is the **blue-print** from which objects are created
- it describes:
  - the **operations** (*methods*) that can be performed on the object
  - the set of **features** (*data members*) of the object



An **object** is and instance of a class.

Structure of a basic Java application

Class

For a minimal application…

- a **class** is needed
- a **main** method is required for execution

```
public class Main {

	public static final int a = 0;
	public final int b = 1;

	public static int c;
	static int d;
	protected int e;
	private int f;

	public static void main(String[] args) {

		int a;
		a = 10;
	}

	private double aDoubleMethod(int a, double b) {
		return a*b;
	}
}
```

Structure of a basic Java application

Class

For a minimal application…

- a **class** is needed
- a **main** method is required for execution

Moreover…

- **variables**
- and **methods** can be declared and defined

```
public class Main {

	public static final int a = 0;
	public final int b = 1;

	public static int c;
	static int d;
	protected int e;
	private int f;

	public static void main(String[] args) {

		int a;
		a = 10;
	}

	private double aDoubleMethod(int a, double b) {
		return a*b;
	}
}
```

Methods

A method is an operation that can be called on objects of the given class

Methods can be declared as follows:

```
modifiers returnType methodName(argumentsType argumentsName) {
   … code …
}
```

```
public static void main(String[] args) {
      int a;
      a = 10;
}
```

The **main** method is a “special” method which is the *entry point* for the execution of the application

Variables

- It's a **memory location** for keeping data, each location is identified by a unique address
- Variables have to be **declared** before their usage to establish the type of data it can handle.
- Data of the corresponding type "*should*" be assigned to the variables
- Variable get default values (0, false, null), using them without **initialisation** leads to compilation errors, unexpected results or exceptions (*more on this later…*)

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

      int a;

      a = 10;
}
```

declaration

initialisation

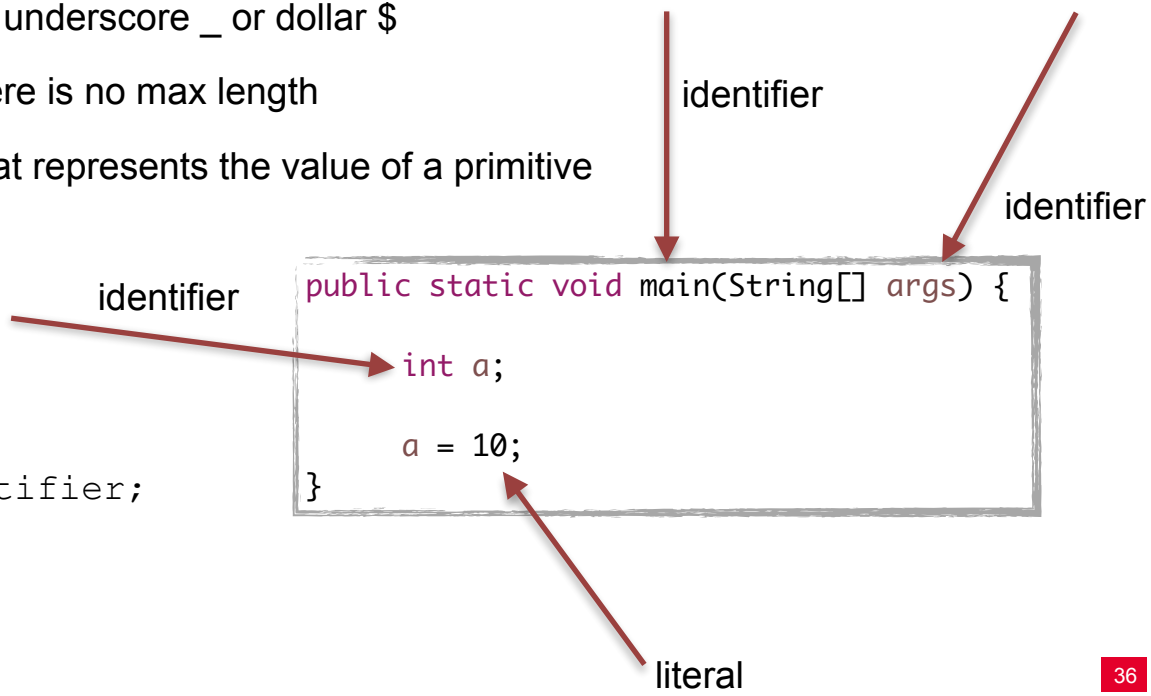
Identifiers and Literals

- Identifiers are the **names** given to variables, methods and classes etc.
- They can begin with a letter, an underscore _ or dollar $
- They are case sensitive and there is no max length
- Literals are the parts of code that represents the value of a primitive

Examples:

- `int _a;`
- `int $c;`
- `int ____2_w;`
- `int this_is_a_long_identifier;`

- `int 2a;`
- `int &b;`
- `int %c;`



Keywords

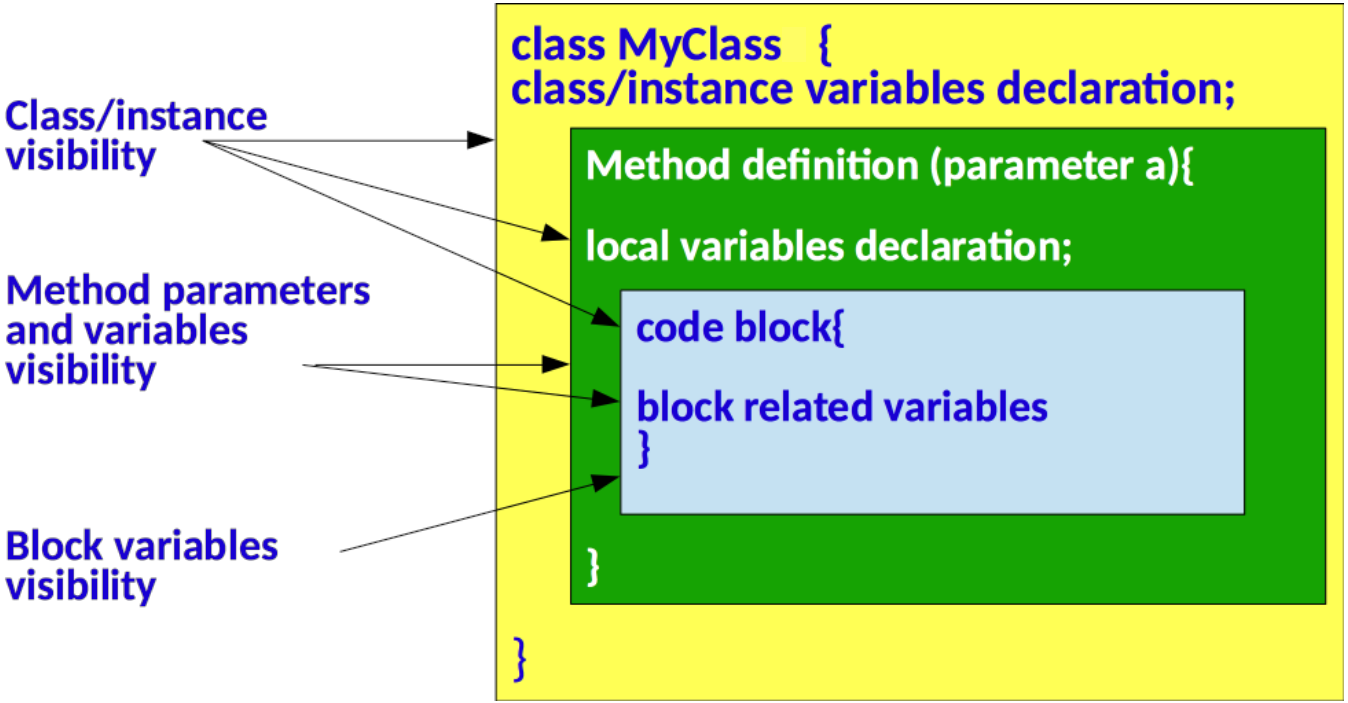
Some keywords exist in Java and thus **cannot** be used as identifiers

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

- const and goto are not used anymore
- *true* and false are boolean literals
- null is also a literal

Scope of variables

Variables are “**visible**” inside the block they have been declared



Types

- Each variable in Java has to be of a given type
- Data can be gathered in two categories:
  - **Primitives**: integers, doubles, chars and booleans
  - **Reference**: pointers to objects and arrays

- References in Java are different from C++: they point to objects but they cannot be manipulated

```
public void primitivesAndReferences() {

      int primitive = 10;

      MyClass object = new MyClass();
}
```

Primitive Types

The following types do not represent objects and are called primitives:

- **byte** : 8 bit integer (values between -128 and 127)
- **short** : 16 bit integer (values between -32768 and 32767)
- **int** : 32 bit integer (values between -2.147.483.648 and 2.147.483.647)
- **long** : 64 bit integer (values between -9.223.372.036.854.775.808 and 9.223.372.036.854.775.807)
- **float** : 32 bit floating point
- **double** : 64 bit floating point
- **char** : 16 bit char (UNICODE) (values between \u0000 = 0 and \uffff = 65535)
- **boolean** : boolean value (true or false)

Default values

Primitives and references get default values

Data Type	Default Value
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
boolean	false
String	null
objects	null

Special characters

Char sequence	Description
\ddd	Octal number
\uxxxx	Exadecimal number
\'	Apostrophy
\”	Double quotes
\\	Backslash
\r	Carriage return
\n	New line
\t	Tab
\b	Backspace

String - Java.lang.String

References

Other than primitives Java provides the type *String* embodied by the `Java.lang.String` class

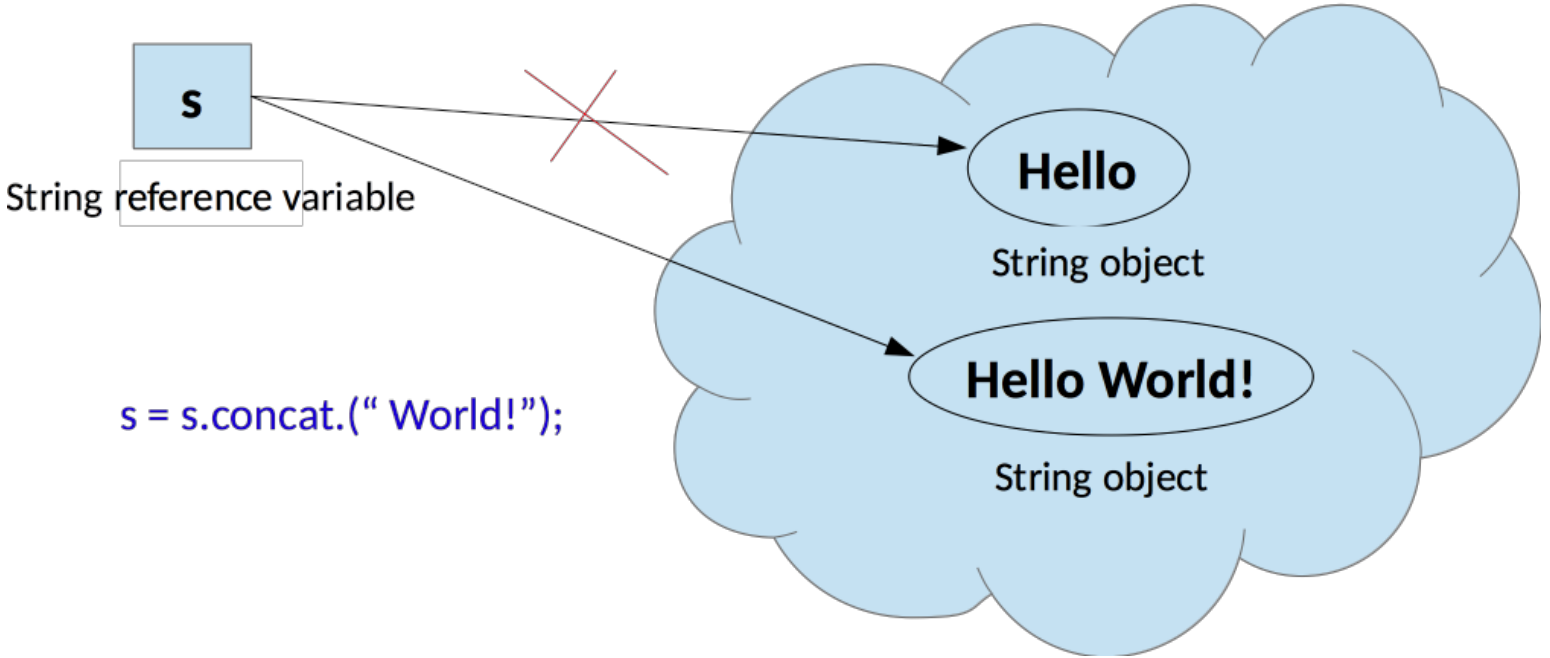
Strings can be created in two ways:

- `String s = new String("hello");`
- `String s = "hello";`

- Strings are **immutable objects**: once created cannot be modified
- But Java provides methods to "edit" strings… what happens then?

String - Java.lang.String

String s = “Hello”;



s = s.concat.(“ World!”);

Arrays

- An array is a container of data of the same type
- Each element can be accessed indicating its position within the array (from 0 to max-1)
- The size of the array has to be provided in the declaration…
- …Or the elements can be specified
- Elements can be retrieved and manipulated using the position

```
double[] doubleArray = new double[5];
double[] doubleArray2 = {1.0, 2.3, 4.6};

double d = doubleArray[3];
doubleArray2[2] = 3.7;
```

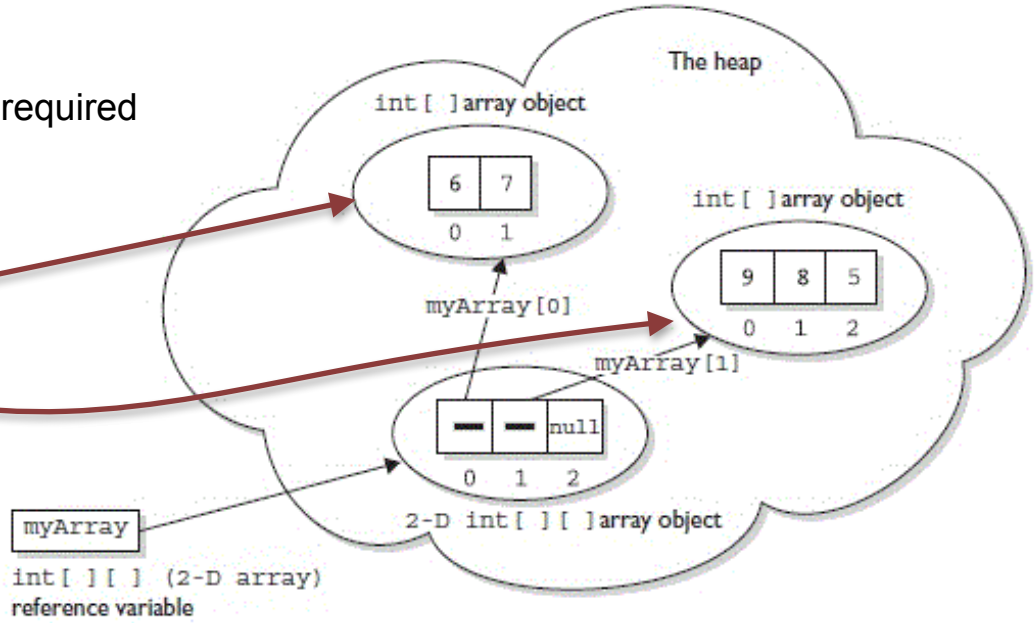
Multidimensional Arrays

Arrays can have more than one dimension

- In the declaration only the first dimension is required

```
int[][] myArray = new int[3][];

int[][] myArray2 = {{6,7},{9,8,5},{}};
```



Creating objects

Objects can be crated starting from the class through their **constructor**

The constructor is a special method with no return type (! not void)

There may be different implementation of the constructor, with different access modifiers

It is implicit if the user doesn't declare it explicitly

It is called using the `new` keyword

```
public static class Person {
      String name;
      int age;

      public Person() {

      }

      public void createPerson() {
            Person p = new Person();
      }
}
```

Creating objects

What happens in memory when an object is created?

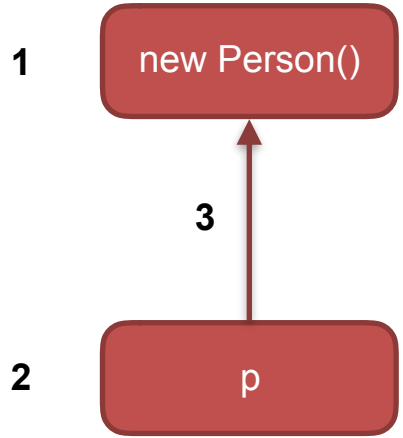
```
public static class Person {
      String name;
      int age;

      public Person() {

      }

      public void createPerson() {
            Person p = new Person();
      }
}
```

2 3 1



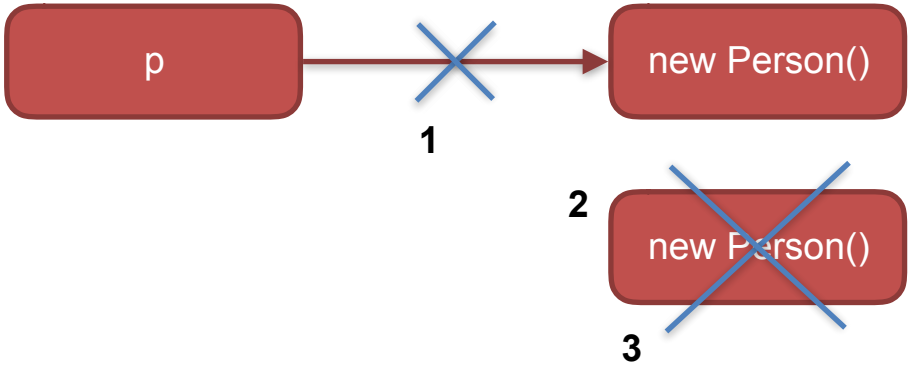
Deleting Objects

Objects do not last in memory forever

Nevertheless it is not possible to destroy objects like in other languages

The **garbage collector** takes care of clearing memory from objects that are no more reachable, i.e there are no more references to them

What happens in memory?

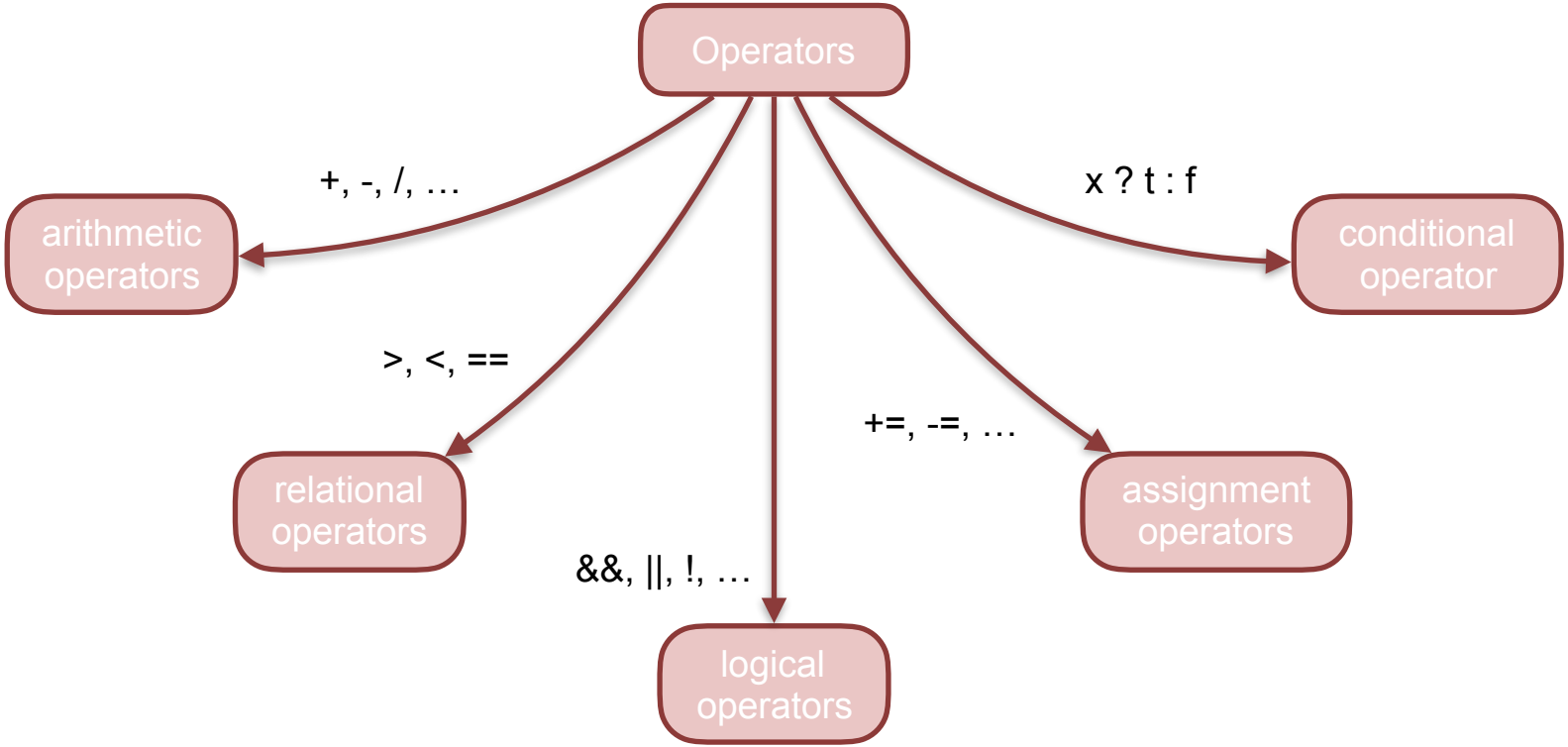



```
…
      Person p = new Person();
      …
      p = null;          1        2
      System.gc(); 3
…
```

Operators

Manipulation of variables and objects

Operators


Operators
+, -, /, …
arithmetic operators
x ? t : f
conditional operator
>, <, ==
relational operators
+=, -=, …
assignment operators
&&, ||, !, …
logical operators

Arithmetic operators

These operators allow to perform basic math operations. Can be classified as:

- **binary** operators
  - **+**
  - **-**
  - *
  - /
  - **%**
- **unary** operators
  - a**++** (“get and increment”)
  - a**--** (“get and decrement”)
  - **++**a (“increment and get”)
  - **--**a (“decrement and get”)

```
int a=10, b=2;
int sum = a+b; // =12
int diff = a-b; // =8
int product = a*b; // =20
int div = a/b; // =5;
int mod = a%b; // =0;
```

for **strings**…

```
String hi = "hi", man = "man";
String hiMan = hi + " " + man;
System.out.println(hiMan); //prints: hi man
```

```
int c = 5, d = 5;
System.out.println(c++ + ", " + ++d); //prints: 5, 6
System.out.println(c + ", " + d); //prints: 6, 6
System.out.println(c-- + ", " + --d); //prints: 6, 5
System.out.println(c + ", " + d); //prints: 5, 5
```

Relational operators

Compare the two operands determining the relationship between them

- op1 **>** op2 returns true if op1 greater of op2
- op1 **>=** op2 returns true if op1 greater or equal to op2
- op1 **<** op2 returns true if op1 smaller of op2
- op1 **<=** op2 returns true if op1 smaller or equal to op2
- op1 **==** op2 returns true if op1 equal to op2
- op1 **!=** op2 returns true if op1 not equal to op2

```
int a=10, b=2;
boolean g = a>b; // true
boolean ge = a>=b; // true
boolean s = a<b; // false
boolean se = a<=b; // false
boolean eq = a==b; // false
boolean ne = a!=b; // true
```

Logical operators

These operators execute logical operations between the operands

- **AND**
  - **&**
  - **&&**

```
boolean andVal = 3>2 & 3>4; // false;
boolean shortAndVal = 3>2 && 3>4; //3>2 not even evaluated
```

- **OR**
  - **|**
  - **||**

```
boolean orVal = 3>4 & 3>2; //true
boolean shortOrVal = 3>4 & 3>2; //3>4 not even evaluated
```

- **NOT**
  - **!** simply returns true when false and viceversa

```
boolean isFalse = !true;
```

Assignment operators

These operators can be considered as a short way to write some expressions

	short expression	expression
• **+=**	op1 += op2	op1 = op1 + op2
• **-=**	op1 -= op2	op1 = op1 - op2
• ***=**	op1 *= op2	op1 = op1 * op2
• **/=**	op1 /= op2	op1 = op1 / op2
• **%=**	op1 %= op2	op1 = op1 % op2

Ternary operator

This operator can be used to return a value depending on a logical condition:

```
variable = (logicalCondition) ? value1 : value2
```

if `logicalCondition` is true then `value1` is assigned to `variable`, `value2` otherwise

It's a good replacement of and if/else structure…