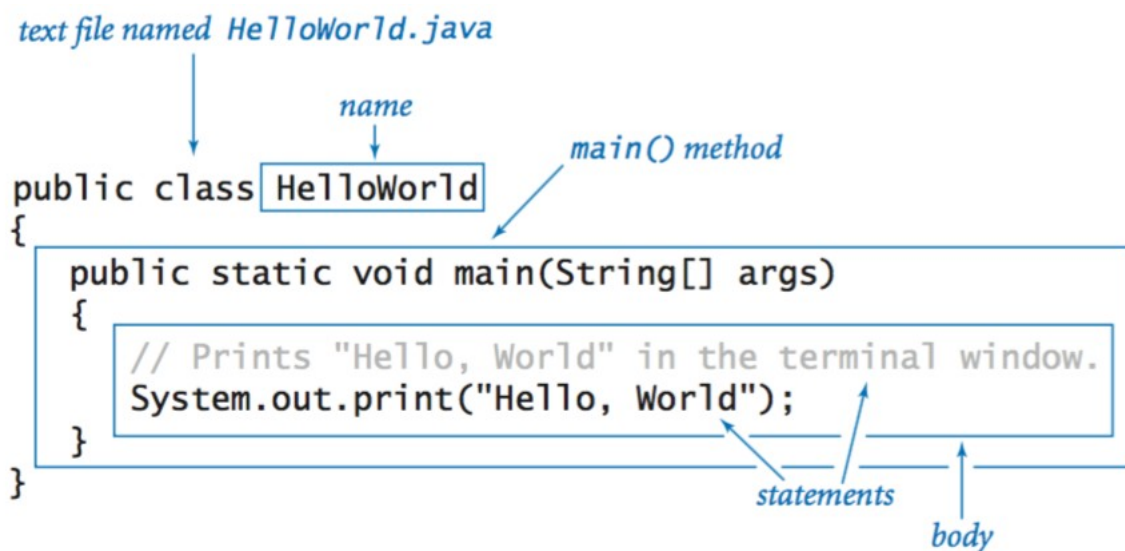


Bibliografia essenziale

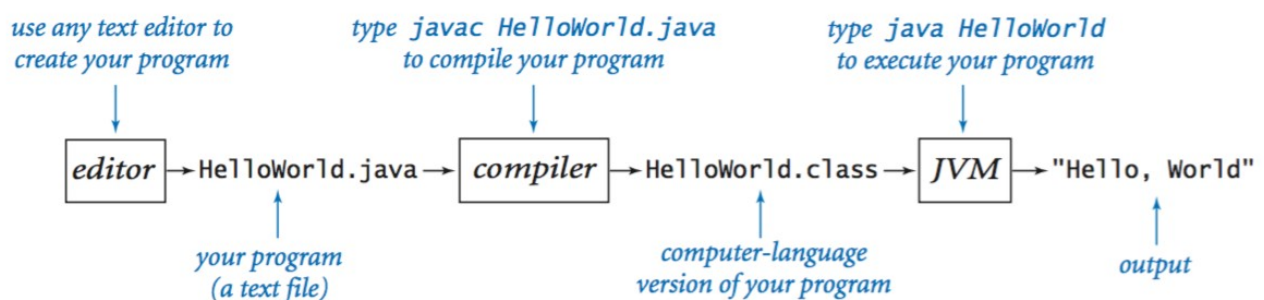
Java SE Documentation – Oracle/Sun (en)
Concetti di informatica e fondamenti di Java - Cay Horstmann (it)
Java SE 8 for the Really Impatient - Cay Horstmann (en)
Manuale di Java 7 – Claudio de Sio – Hoepli Informatica (it)
Thinking in Java Vol. 1 I Fondamenti – Bruce Eckel (en)
Java Fondamenti di Programmazione – Deitel&Deitel (it)
Corso Java – html.it (it)

1 - Introduzione a Java 7

HelloWorld con Java



Il processo di compilazione



2 - Componenti fondamentali di un programma Java

Le basi della programmazione object oriented: classi e oggetti

I metodi in Java

3 - Identificatori, tipi di dati

<i>type</i>	<i>set of values</i>	<i>common operators</i>	<i>sample literal values</i>
int	integers	+ - * / %	99 12 2147483647
double	floating-point numbers	+ - * /	3.14 2.5 6.022e23
boolean	boolean values	&& !	true false
char	characters		'A' '1' '%' '\n'
String	sequences of characters	+	"AB" "Hello" "2.5"

<i>values</i>	integers between -2^{31} and $+2^{31}-1$					
<i>typical literals</i>	1234 99 0 1000000					
<i>operations</i>	<i>sign</i>	<i>add</i>	<i>subtract</i>	<i>multiply</i>	<i>divide</i>	<i>remainder</i>
<i>operators</i>	+ -	+	-	*	/	%

<i>expression</i>	<i>value</i>	<i>comment</i>
99	99	<i>integer literal</i>
+99	99	<i>positive sign</i>
-99	-99	<i>negative sign</i>
5 + 3	8	<i>addition</i>
5 - 3	2	<i>subtraction</i>
5 * 3	15	<i>multiplication</i>
5 / 3	1	<i>no fractional part</i>
5 % 3	2	<i>remainder</i>
1 / 0		<i>run-time error</i>
3 * 5 - 2	13	<i>* has precedence</i>
3 + 5 / 2	5	<i>/ has precedence</i>
3 - 5 - 2	-4	<i>left associative</i>
(3 - 5) - 2	-4	<i>better style</i>
3 - (5 - 2)	0	<i>unambiguous</i>

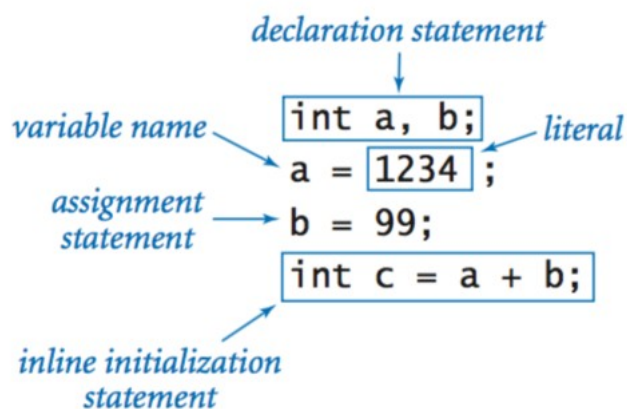
Booleans

<i>values</i>	<i>true or false</i>		
<i>literals</i>	<code>true</code>	<code>false</code>	
<i>operations</i>	<code>and</code>	<code>or</code>	<code>not</code>
<i>operators</i>	<code>&&</code>	<code> </code>	<code>!</code>

4 - Operatori e gestione del flusso di esecuzione

OPERATORI DI BASE

Operatore d'assegnazione



<i>op</i>	<i>meaning</i>	<i>true</i>	<i>false</i>
<code>==</code>	<i>equal</i>	<code>2 == 2</code>	<code>2 == 3</code>
<code>!=</code>	<i>not equal</i>	<code>3 != 2</code>	<code>2 != 2</code>
<code><</code>	<i>less than</i>	<code>2 < 13</code>	<code>2 < 2</code>
<code><=</code>	<i>less than or equal</i>	<code>2 <= 2</code>	<code>3 <= 2</code>
<code>></code>	<i>greater than</i>	<code>13 > 2</code>	<code>2 > 13</code>
<code>>=</code>	<i>greater than or equal</i>	<code>3 >= 2</code>	<code>2 >= 3</code>

non-negative discriminant?

`(b*b - 4.0*a*c) >= 0.0`

beginning of a century?

`(year % 100) == 0`

legal month?

`(month >= 1) && (month <= 12)`

Printing

<code>void System.out.print(String s)</code>	<i>print s</i>
<code>void System.out.println(String s)</code>	<i>print s, followed by a newline</i>
<code>void System.out.println()</code>	<i>print a newline</i>

Parsing command-line arguments.

<code>int Integer.parseInt(String s)</code>	<i>convert s to an int value</i>
<code>double Double.parseDouble(String s)</code>	<i>convert s to a double value</i>
<code>long Long.parseLong(String s)</code>	<i>convert s to a long value</i>

Concatenazione di stringhe con +

Math library.

`public class Math`

<code>double abs(double a)</code>	<i>absolute value of a</i>
<code>double max(double a, double b)</code>	<i>maximum of a and b</i>
<code>double min(double a, double b)</code>	<i>minimum of a and b</i>
<code>double sin(double theta)</code>	<i>sine of theta</i>
<code>double cos(double theta)</code>	<i>cosine of theta</i>
<code>double tan(double theta)</code>	<i>tangent of theta</i>
<code>double toRadians(double degrees)</code>	<i>convert angle from degrees to radians</i>
<code>double toDegrees(double radians)</code>	<i>convert angle from radians to degrees</i>
<code>double exp(double a)</code>	<i>exponential (e^a)</i>
<code>double log(double a)</code>	<i>natural log ($\log_e a$, or $\ln a$)</i>
<code>double pow(double a, double b)</code>	<i>raise a to the bth power (a^b)</i>
<code>long round(double a)</code>	<i>round a to the nearest integer</i>
<code>double random()</code>	<i>random number in [0, 1)</i>
<code>double sqrt(double a)</code>	<i>square root of a</i>
<code>double E</code>	<i>value of e (constant)</i>
<code>double PI</code>	<i>value of π (constant)</i>

Java library calls.

<i>method call</i>	<i>library</i>	<i>return type</i>	<i>value</i>
<code>Integer.parseInt("123")</code>	Integer	int	123
<code>Double.parseDouble("1.5")</code>	Double	double	1.5
<code>Math.sqrt(5.0*5.0 - 4.0*4.0)</code>	Math	double	3.0
<code>Math.log(Math.E)</code>	Math	double	1.0
<code>Math.random()</code>	Math	double	random in [0, 1)
<code>Math.round(3.14159)</code>	Math	long	3
<code>Math.max(1.0, 9.0)</code>	Math	double	9.0

Type conversion.

<i>expression</i>	<i>expression type</i>	<i>expression value</i>
<code>(1 + 2 + 3 + 4) / 4.0</code>	double	2.5
<code>Math.sqrt(4)</code>	double	2.0
<code>"1234" + 99</code>	String	"123499"
<code>11 * 0.25</code>	double	2.75
<code>(int) 11 * 0.25</code>	double	2.75
<code>11 * (int) 0.25</code>	int	0
<code>(int) (11 * 0.25)</code>	int	2
<code>(int) 2.71828</code>	int	2
<code>Math.round(2.71828)</code>	long	3
<code>(int) Math.round(2.71828)</code>	int	3
<code>Integer.parseInt("1234")</code>	int	1234

Floating-point numbers.

<i>values</i>	real numbers (specified by IEEE 754 standard)			
<i>typical literals</i>	3.14159	6.022e23	2.0	1.4142135623730951
<i>operations</i>	<i>add</i>	<i>subtract</i>	<i>multiply</i>	<i>divide</i>
<i>operators</i>	+	-	*	/

If and if-else statements.

<i>absolute value</i>	<code>if (x < 0) x = -x;</code>
<i>put the smaller value in x and the larger value in y</i>	<code>if (x > y) { int t = x; x = y; y = t; }</code>
<i>maximum of x and y</i>	<code>if (x > y) max = x; else max = y;</code>
<i>error check for division operation</i>	<code>if (den == 0) System.out.println("Division by zero"); else System.out.println("Quotient = " + num/den);</code>
<i>error check for quadratic formula</i>	<code>double discriminant = b*b - 4.0*c; if (discriminant < 0.0) { System.out.println("No real roots"); } else { System.out.println((-b + Math.sqrt(discriminant))/2.0); System.out.println((-b - Math.sqrt(discriminant))/2.0); }</code>

Nested if-else statement.

```
if      (income <      0) rate = 0.00;  
else if (income <  8925) rate = 0.10;  
else if (income < 36250) rate = 0.15;  
else if (income < 87850) rate = 0.23;  
else if (income < 183250) rate = 0.28;  
else if (income < 398350) rate = 0.33;  
else if (income < 400000) rate = 0.35;  
else                                rate = 0.396;
```

While loops.

initialization is a separate statement
`int power = 1;`
loop-continuation condition
`while (power <= n/2)`
braces are optional when body is a single statement
`{`
`power = 2*power;`
`}`
body

For loops.

initialize another variable in a separate statement
`int power = 1;`
declare and initialize a loop control variable
`for (int i = 0;`
loop-continuation condition
`i <= n;`
increment
`i++)`
`{`
`System.out.println(i + " " + power);`
`power = 2*power;`
`}`
body

Loops

<i>compute the largest power of 2 less than or equal to n</i>	<pre>int power = 1; while (power <= n/2) power = 2*power; System.out.println(power);</pre>
<i>compute a finite sum (1 + 2 + ... + n)</i>	<pre>int sum = 0; for (int i = 1; i <= n; i++) sum += i; System.out.println(sum);</pre>
<i>compute a finite product (n! = 1 × 2 × ... × n)</i>	<pre>int product = 1; for (int i = 1; i <= n; i++) product *= i; System.out.println(product);</pre>
<i>print a table of function values</i>	<pre>for (int i = 0; i <= n; i++) System.out.println(i + " " + 2*Math.PI*i/n);</pre>
<i>compute the ruler function (see PROGRAM 1.2.1)</i>	<pre>String ruler = "1"; for (int i = 2; i <= n; i++) ruler = ruler + " " + i + " " + ruler; System.out.println(ruler);</pre>

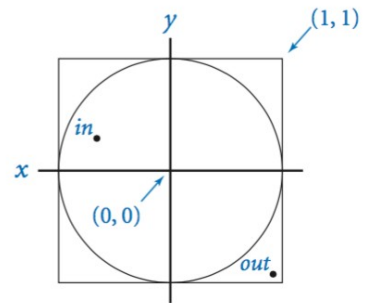
Break statement.

```
int factor;
for (factor = 2; factor <= n/factor; factor++)
    if (n % factor == 0) break;

if (factor > n/factor)
    System.out.println(n + " is prime");
```


Do-while loop.

```
do
{ // Scale x and y to be random in (-1, 1).
  x = 2.0*Math.random() - 1.0;
  y = 2.0*Math.random() - 1.0;
} while (Math.sqrt(x*x + y*y) > 1.0);
```



Switch statement.

```
switch (day) {
  case 0: System.out.println("Sun"); break;
  case 1: System.out.println("Mon"); break;
  case 2: System.out.println("Tue"); break;
  case 3: System.out.println("Wed"); break;
  case 4: System.out.println("Thu"); break;
  case 5: System.out.println("Fri"); break;
  case 6: System.out.println("Sat"); break;
}
```

Arrays

a	a[0]
	a[1]
	a[2]
	a[3]
	a[4]
	a[5]
	a[6]
	a[7]

Inline array initialization.

```
String[] SUITS = { "Clubs", "Diamonds", "Hearts", "Spades" };
```

```
String[] RANKS = {
  "2", "3", "4", "5", "6", "7", "8", "9", "10",
  "Jack", "Queen", "King", "Ace"
};
```

Typical array-processing code.

<i>create an array with random values</i>	<pre>double[] a = new double[n]; for (int i = 0; i < n; i++) a[i] = Math.random();</pre>
<i>print the array values, one per line</i>	<pre>for (int i = 0; i < n; i++) System.out.println(a[i]);</pre>
<i>find the maximum of the array values</i>	<pre>double max = Double.NEGATIVE_INFINITY; for (int i = 0; i < n; i++) if (a[i] > max) max = a[i];</pre>
<i>compute the average of the array values</i>	<pre>double sum = 0.0; for (int i = 0; i < n; i++) sum += a[i]; double average = sum / n;</pre>
<i>reverse the values within an array</i>	<pre>for (int i = 0; i < n/2; i++) { double temp = a[i]; a[i] = a[n-1-i]; a[n-i-1] = temp; }</pre>
<i>copy sequence of values to another array</i>	<pre>double[] b = new double[n]; for (int i = 0; i < n; i++) b[i] = a[i];</pre>

INTRODUZIONE ALLA LIBRERIA STANDARD

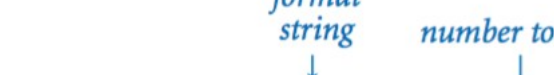
public class StdOut

void print(String s)	<i>print s to standard output</i>
void println(String s)	<i>print s and a newline to standard output</i>
void println()	<i>print a newline to standard output</i>
void printf(String format, ...)	<i>print the arguments to standard output, as specified by the format string format</i>

Il comando import

La classe String

La documentazione della libreria standard di Java



The diagram illustrates the components of the format string `"%7.5f"` in the `StdOut.printf` call. Arrows point from descriptive labels to the corresponding parts of the format string:

- format string**: Points to the entire `"%7.5f"` sequence.
- number to print**: Points to the `Math.PI` argument.
- field width**: Points to the `7` in the format string.
- precision**: Points to the `5` in the format string.
- conversion specification**: Points to the `f` in the format string.

<i>type</i>	<i>code</i>	<i>typical literal</i>	<i>sample format strings</i>	<i>converted string values for output</i>
int	d	512	"%14d" "%-14d"	"512"
double	f	1595.1680010754388	"%14.2f"	"1595.17"
	e		"%.7f"	"1595.1680011"
			"%14.4e"	"1.5952e+03"
String	s	"Hello, World"	"%14s" "%-14s" "%-14.5s"	" Hello, World" "Hello, World " "Hello "
boolean	b	true	"%b"	"true"

standard input library.

public class StdIn

methods for reading individual tokens from standard input

boolean isEmpty()	<i>is standard input empty (or only whitespace)?</i>
int readInt()	<i>read a token, convert it to an int, and return it</i>
double readDouble()	<i>read a token, convert it to a double, and return it</i>
boolean readBoolean()	<i>read a token, convert it to a boolean, and return it</i>
String readString()	<i>read a token and return it as a String</i>

methods for reading characters from standard input

boolean hasNextChar()	<i>does standard input have any remaining characters?</i>
char readChar()	<i>read a character from standard input and return it</i>

methods for reading lines from standard input

boolean hasNextLine()	<i>does standard input have a next line?</i>
String readLine()	<i>read the rest of the line and return it as a String</i>

methods for reading the rest of standard input

int[] readAllInts()	<i>read all remaining tokens and return them as an int array</i>
double[] readAllDoubles()	<i>read all remaining tokens and return them as a double array</i>
boolean[] readAllBooleans()	<i>read all remaining tokens and return them as a boolean array</i>
String[] readAllStrings()	<i>read all remaining tokens and return them as a String array</i>
String[] readAllLines()	<i>read all remaining lines and return them as a String array</i>
String readAll()	<i>read the rest of the input and return it as a String</i>

standard drawing library.

public class StdDraw

drawing commands

```
void line(double x0, double y0, double x1, double y1)
void point(double x, double y)
void circle(double x, double y, double radius)
void filledCircle(double x, double y, double radius)
void square(double x, double y, double radius)
void filledSquare(double x, double y, double radius)
void rectangle(double x, double y, double r1, double r2)
void filledRectangle(double x, double y, double r1, double r2)
void polygon(double[] x, double[] y)
void filledPolygon(double[] x, double[] y)
void text(double x, double y, String s)
void picture(double x, double y, String filename)
```

control commands

void setXscale(double x0, double x1)	<i>reset x-scale to (x0, x1)</i>
void setYscale(double y0, double y1)	<i>reset y-scale to (y0, y1)</i>
void setPenRadius(double radius)	<i>set pen radius to radius</i>
void setPenColor(Color color)	<i>set pen color to color</i>
void setFont(Font font)	<i>set text font to font</i>
void setCanvasSize(int w, int h)	<i>set canvas size to w-by-h</i>
void clear(Color color)	<i>clear the canvas to color color</i>
void show(int dt)	<i>show and pause dt milliseconds</i>
void save(String filename)	<i>save to a .jpg or .png file</i>

Gli array in Java

Array Quick Examples

```
String personOne = "Mauro";
String personTwo = "Paolo"
;
String [] array = {personOne, personTwo};

for (String person : array) {
Log.d("person:",person);
}

String mauro = array[0];
```


Dichiarazione

Creazione

Inizializzazione

5 - Programmazione ad oggetti utilizzando Java: incapsulamento ed ereditarietà

Using an object.

declare a variable (object name)

```
String s;
```

invoke a constructor to create an object

```
s1 = new String("Hello, World");
```

object name

```
char c = s.charAt(4);
```

invoke an instance method that operates on the object's value

Creating an object.

Variabili di istanza

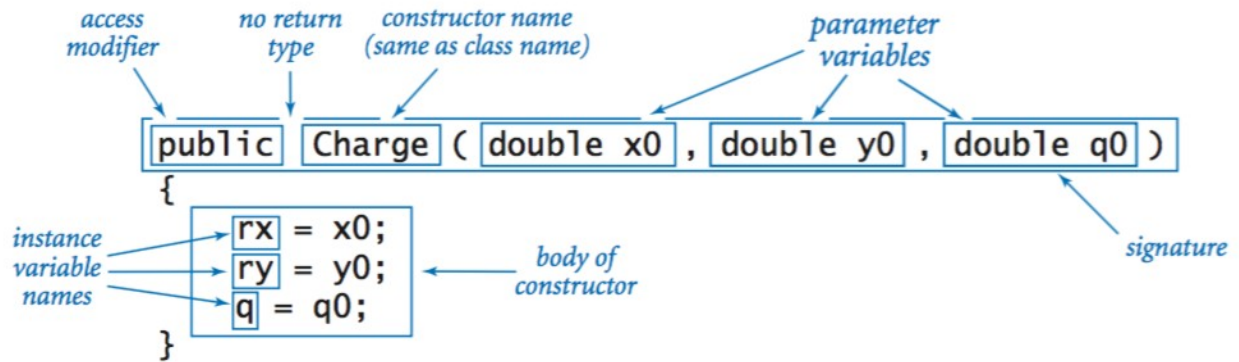
```
public class Charge
{
    private final double rx, ry;
    private final double q;
    .
    .
}
```

instance variable declarations

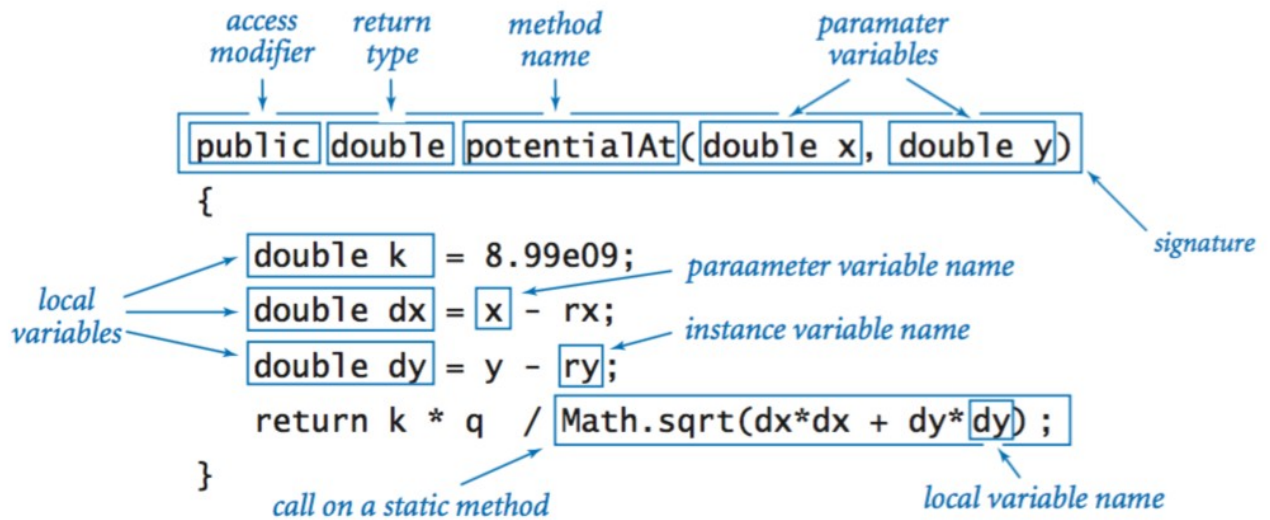
access modifiers

Instance variables

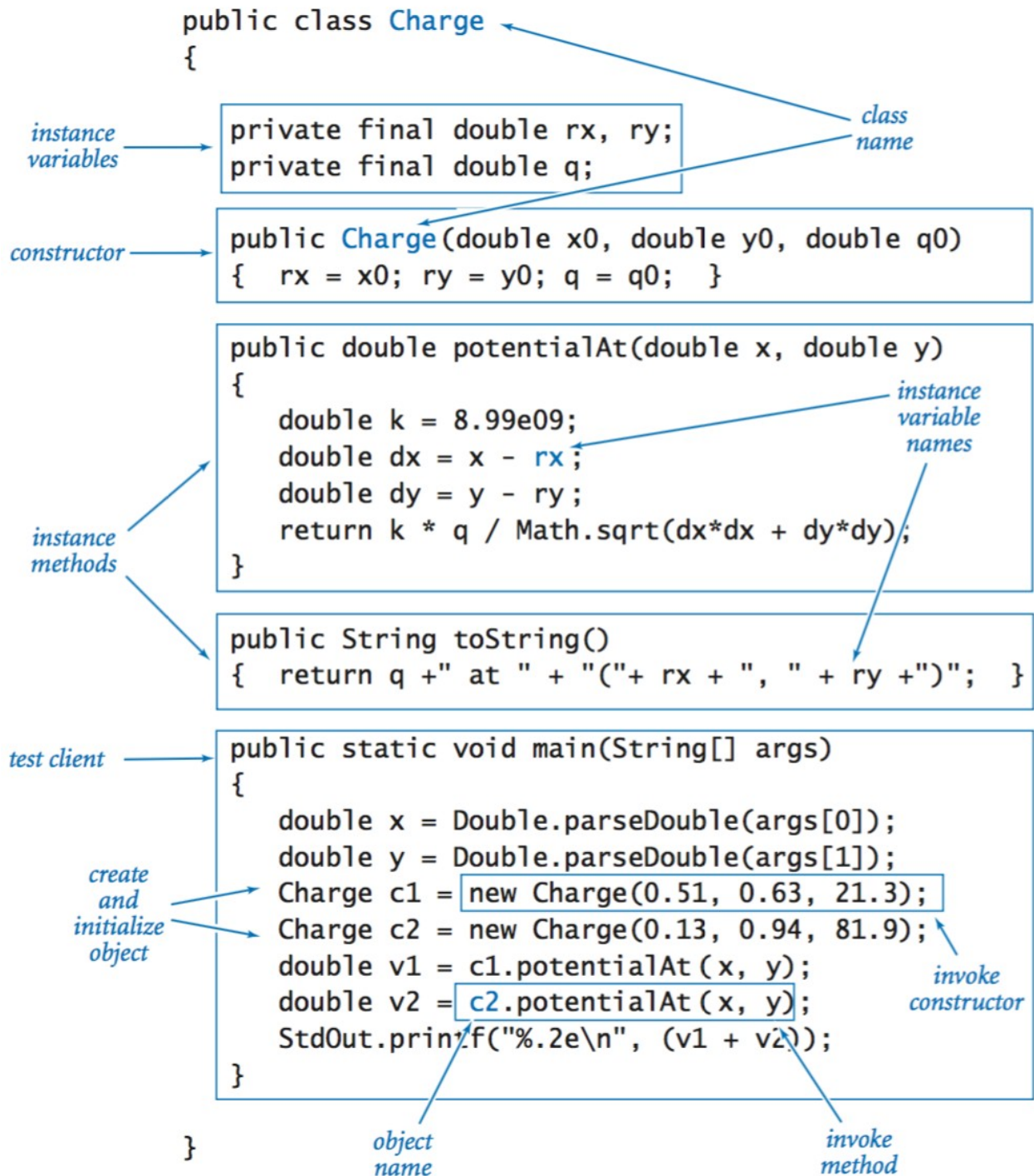
Costruttori



metodi



Classi



Object-oriented libraries.

client

```
Charge c1 = new Charge(0.51, 0.63, 21.3);  
  
c1.potentialAt(x, y)
```

*creates objects
and invokes methods*

API

```
public class Charge  
{  
    Charge(double x0, double y0, double q0)  
    double potentialAt(double x, double y) potential at (x, y)  
                                             due to charge  
    String toString() string  
                      representation  
}
```

*defines signatures
and describes methods*

implementation

```
public class Charge  
{  
    private final double rx, ry;  
    private final double q;  
  
    public Charge(double x0, double y0, double q0)  
    { ... }  
  
    public double potentialAt(double x, double y)  
    { ... }  
  
    public String toString()  
    { ... }  
}
```

*defines instance variables
and implements methods*

Java's String data type.

```
public class String
```

<code>String(String s)</code>	<i>create a string with the same value as s</i>
<code>int length()</code>	<i>number of characters</i>
<code>char charAt(int i)</code>	<i>the character at index i</i>
<code>String substring(int i, int j)</code>	<i>characters at indices i through (j-1)</i>
<code>boolean contains(String substring)</code>	<i>does this string contain substring?</i>
<code>boolean startsWith(String pre)</code>	<i>does this string start with pre?</i>
<code>boolean endsWith(String post)</code>	<i>does this string end with post?</i>
<code>int indexOf(String pattern)</code>	<i>index of first occurrence of pattern</i>
<code>int indexOf(String pattern, int i)</code>	<i>index of first occurrence of pattern after i</i>
<code>String concat(String t)</code>	<i>this string with t appended</i>
<code>int compareTo(String t)</code>	<i>string comparison</i>
<code>String toLowerCase()</code>	<i>this string, with lowercase letters</i>
<code>String toUpperCase()</code>	<i>this string, with uppercase letters</i>
<code>String replaceAll(String a, String b)</code>	<i>this string, with as replaced by bs</i>
<code>String[] split(String delimiter)</code>	<i>strings between occurrences of delimiter</i>
<code>boolean equals(Object t)</code>	<i>is this string's value the same as t's?</i>
<code>int hashCode()</code>	<i>an integer hash code</i>

Programmazione ad oggetti utilizzando Java: polimorfismo

Polimorfismo

Overload

'overload che consente di definire in una stessa classe più metodi aventi lo stesso nome, ma che differiscano nella firma, cioè nella sequenza dei tipi dei parametri formali.

E' compito del compilatore determinare quale dei metodi "overloadati" dovrà essere invocato, in base al numero e al tipo dei parametri attuali.

```
public class OperazioniSuNumeri {  
    public int somma(int x, int y) {  
        return x+y;  
    }  
}
```



```
public float somma(float x, float y) {  
    return x+y;  
}  
}
```

Override

L'override consente di ridefinire un metodo in una sottoclasse: una vera e propria riscrittura di un certo metodo di una classe ereditata.

Non esiste override senza ereditarietà; sia il metodo originale che quello che lo ridefinisce devono necessariamente avere la stessa firma.

Sarà compito dell'interprete, durante l'esecuzione del programma, determinare quale dei due metodi dovrà essere eseguito.

Override e classe Object: metodi `toString()`, `clone()`, `equals()` e `hashCode()`

Il metodo `toString()` si usa per convenzione per generare una stringa che rappresenti l'oggetto. La definizione default del metodo nella classe `Object` restituisce una stringa contenente informazioni sul reference. Il metodo deve essere ridefinito in ogni classe che lo usa per ottenere un risultato significativo.

Caratteristiche avanzate del linguaggio

8 Costruttori e polimorfismo

8 Overload dei costruttori

8 Override dei costruttori

8 Costruttori ed ereditarietà

8 super: un "super reference"

8 super e i costruttori

8 Altri componenti di un'applicazione Java: classi innestate e anonime

8 Classi innestate: introduzione e storia

8 Classe innestata: definizione

8 Classi innestate: proprietà

8 Classi anonime: definizione

9 - Modificatori, package e interfacce

MODIFICATORE	CLASSE	ATTRIBUTO	METODO	COSTRUTTORE	BLOCCO DI CODICE
public	sì	sì	sì	sì	no
protected	no	sì	sì	sì	no
(default)	sì	sì	sì	sì	sì
private	no	sì	sì	sì	no
abstract	sì	no	sì	no	no
final	sì	sì	sì	no	no
native	no	no	sì	no	no
static	no	sì	sì	no	sì
strictfp	sì	no	sì	no	no
synchronized	no	no	sì	no	no
volatile	no	sì	no	no	no
transient	no	sì	no	no	no

MODIFICATORE	STESSA CLASSE	STESSO PACKAGE	SOTTOCLASSE	OVUNQUE
public	sì	sì	sì	sì
protected	sì	sì	sì	no
(default)	sì	sì	no	no
private	sì	no	no	no

- **Il modificatore final**

- una variabile dichiarata **final** diventa una **costante**;
- un metodo dichiarato **final** non può essere riscritto in una sottoclasse (non è possibile applicare l'**override**);
- una classe dichiarata **final** non può essere **estesa**.

- **Il modificatore static**

- “condiviso da tutte le istanze della classe”, oppure “della classe”.
- Metodi statici
 - p.es metodo sqrt() della classe Math
- Variabili statiche (di classe)
 - Una variabile **statica**, essendo condivisa da tutte le istanze della classe, assumerà lo stesso valore per ogni oggetto di una classe. se un'istanza modifica la variabile statica, essa risulterà modificata anche

relativamente all'altra istanza. P. es. Il record di un gioco, o un contatore globale.

- **Il modificatore abstract**

- Metodi astratti
 - Un metodo astratto non implementa un proprio blocco di codice e quindi il suo comportamento. In pratica, un metodo astratto non definisce parentesi graffe, ma termina con un punto e virgola. Potrà essere definito solamente all'interno di una classe astratta.
- Classi astratte
 - Una classe dichiarata astratta non può essere istanziata. Una classe astratta è che “**obbliga**” le sue sottoclassi a implementare un comportamento.

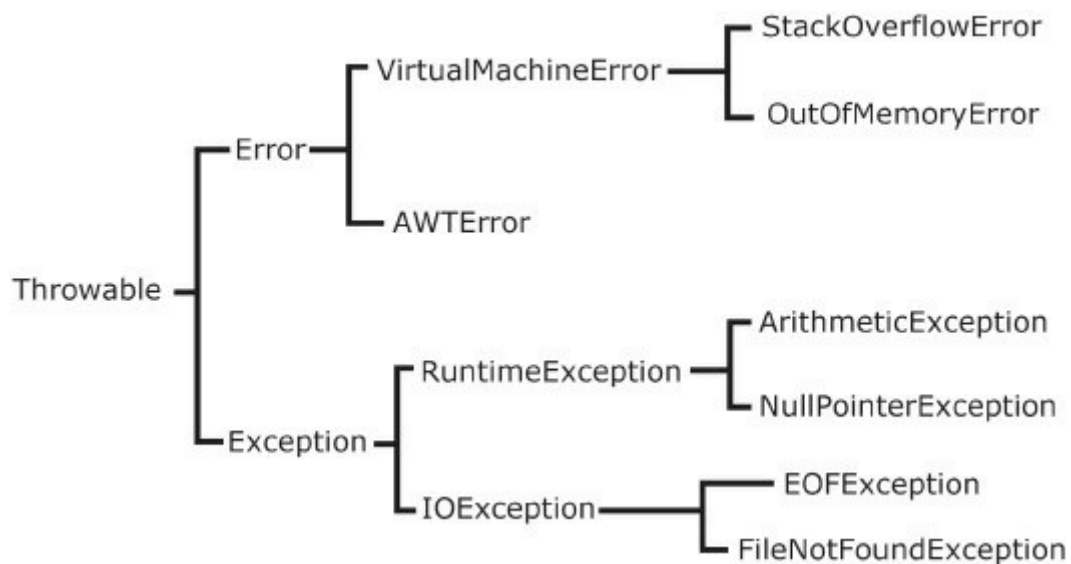
- **Interfacce**

- Un'interfaccia è un'evoluzione del concetto di classe astratta. Per definizione un'interfaccia possiede
 - tutti i metodi dichiarati **public** e **abstract**
 - tutte le variabili dichiarate **public**, **static** e **final**.
 - siccome un'interfaccia non può dichiarare metodi non astratti, non c'è bisogno di marcarli con **abstract** (e con **public**)
- sia le classi astratte sia le interfacce risiede nel fatto che esse possono “obbligare” le sottoclassi a implementare comportamenti. Una classe che eredita un metodo astratto, infatti, deve fare override del metodo ereditato oppure essere dichiarata astratta.
- Possiamo simulare l'ereditarietà multipla solo con l'utilizzo di interfacce.
- Un'interfaccia non può dichiarare né variabili né metodi concreti, ma solo costanti statiche e pubbliche e metodi astratti.
- È possibile dichiarare una classe astratta anche senza metodi astratti. Dichiararla astratta implica comunque che non possa essere istanziata.
- **Tipi enumerazioni**
 - una struttura dati che si aggiunge alle classi, alle interfacce e alle annotazioni
 - da Java 5 una nuova parola chiave: **enum**
 - strutture dati somiglianti alle classi, ma con proprietà particolari
 - Un'enum non si può istanziare come una classe,
 - le uniche istanze che esistono sono proprio i suoi elementi

- Tutte le istanze di un'enumerazione sono implicitamente dichiarate **public**, **static** e **final**.
- In una enum è possibile anche creare **costruttori** che sono implicitamente dichiarati **private** e non è possibile utilizzarli se non nell'ambito dell'enumerazione stessa.

Eccezioni

- Comprendere le varie tipologie di eccezioni, errori e asserzioni
- Saper gestire le varie tipologie di eccezioni con i blocchi **try-catch**
- Saper creare tipi di eccezioni personalizzate e gestire il meccanismo di propagazione con le parole chiave **throw** e **throws**



NullPointerException: probabilmente la più frequente tra le eccezioni. Viene lanciata dalla JVM quando per esempio viene chiamato un metodo su di un reference che invece punta a null.

ArrayIndexOutOfBoundsException: questa eccezione viene lanciata quando si prova ad accedere a un indice di un array troppo alto.

ClassCastException: eccezione particolarmente insidiosa. Viene lanciata al runtime quando si prova a effettuare un cast verso un tipo di classe sbagliato.

Se si utilizzano altri **package** come **java.io**, bisognerà gestire spesso le eccezioni come **IOException** e le sue sottoclassi (**FileNotFoundException**, **EOFException** ecc.).

Stesso discorso con la libreria **java.sql** e l'eccezione **SQLException**,

la vediamo **in android** → **ListActivity** → **Class.forName**

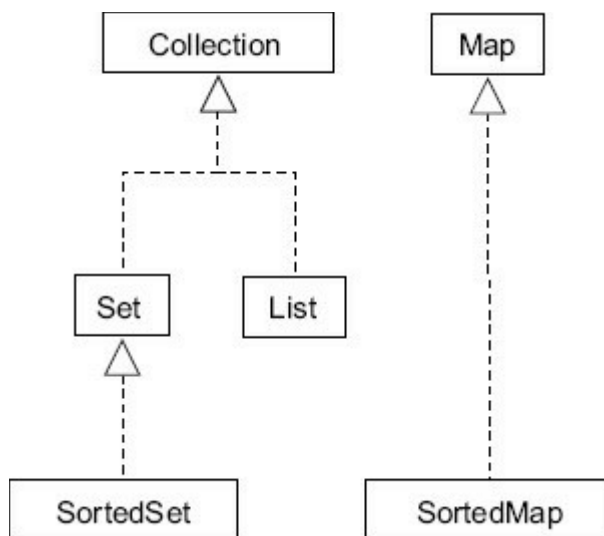
Gestione dei thread

La vediamo solo **in android** → **splashscreen**

Le librerie alla base del linguaggio: java.lang e java.util

Package java.util

Framework Collections



Implementazioni di Map e SortedMap

Una tipica implementazione di una mappa è l'Hashtable. Questa classe permette di associare a ogni elemento della collezione una chiave univoca.

Si aggiungono elementi mediante il metodo `put(Object key, Object value)` e si recuperano tramite il metodo `get(Object key)`.

La classe `HashMap` è del tutto simile a `Hashtable`, ma è molto più usata perché più performante.

```
HashMap<String, String> map = new HashMap<>();
map.put("personOne", "Maurino");
// aggiorna il valore esistente, se esiste
map.put("personOne", "Mauro");
map.put("personTwo", "Paolo");
// rimuovi il valore per la key
map.remove("personOne");
// iterazione dei valori
for (Map.Entry<String, String> personEntry : map.entrySet()) {
    Log.d(personEntry.getKey(),
        personEntry.getValue());
}
```


Implementazioni di Set e SortedSet

Un'implementazione di Set è HashSet, mentre un'implementazione di SortedSet è TreeSet.

HashSet è più performante rispetto a TreeSet ma non gestisce l'ordinamento. Entrambe queste classi non ammettono elementi duplicati.

```
TreeSet set = new TreeSet();  
set.add("c");  
set.add("a");  
set.add("b");  
set.add("b");  
Iterator iter = set.iterator();  
while (iter.hasNext()) {  
    System.out.println(iter.next());  
}
```

Implementazioni di List

Le principali implementazioni di List sono ArrayList, Vector e LinkedList. Vector e ArrayList hanno la stessa funzionalità ma quest'ultima è più performante perché non è sincronizzata. ArrayList ha prestazioni nettamente superiori anche a LinkedList, che conviene utilizzare solo per gestire collezioni di tipo "coda".

Date, orari e valute

```
Date date = new Date();  
SimpleDateFormat df = new SimpleDateFormat("dd-MM-yyyy");  
System.out.println(df.format(date));
```

—

```
Locale locale = Locale.ITALY;  
DateFormat formatter =  
    DateFormat.getInstance(DateFormat.SHORT, locale);  
String s = formatter.format(new Date());  
s = DateFormat.getInstance(DateFormat.MEDIUM,  
    locale).format(new Date());  
System.out.println(s);
```

```

—
double number = 55667788.12345;
Locale localeUsa = Locale.US;
Locale localeIta = Locale.ITALY;
NumberFormat usaFormat = NumberFormat.getInstance(localeUsa);
String usaNumber = usaFormat.format(number);
System.out.println(localeUsa.getDisplayCountry() + " " + usaNumber);
NumberFormat itaFormat = NumberFormat.getInstance(localeIta);
String itaNumber = itaFormat.format(number);
System.out.println(localeIta.getDisplayCountry() + " " + itaNumber);
--
public int getNumeroSettimana(Date date){
    Calendar calendar = new GregorianCalendar();
    calendar.setTime(date);
    int settimana = calendar.get(Calendar.WEEK_OF_YEAR);
    return settimana;
}

```

```

—
Calendar cal = new
GregorianCalendar(TimeZone.getTimeZone("Europe/London"));
int hourOfDay = cal.get(Calendar.HOUR_OF_DAY);

```

La classe StringTokenizer

StringTokenizer è una semplice classe che permette di separare i contenuti di una stringa in più parti (“token”).

```

StringTokenizer st = new StringTokenizer("questo è un test");
while (st.hasMoreTokens()) {
    System.out.println(st.nextToken());
}

```

stampato fino a qui il 15/2/2016

13 - Comunicare con Java: input, output e networking

13 Introduzione all'input-output

13 NIO 2.0 (New Input Output aggiornato a Java 7)

14 - Java e la gestione dei dati: supporto a SQL e XML

14 Introduzione a JDBC

14 Le basi di JDBC

14 Implementazione del vendor (Driver JDBC)

14 Implementazione dello sviluppatore (Applicazione JDBC)

15 - Interfacce grafiche (GUI) con AWT, Applet e Swing

15 Introduzione alla Graphical User Interface (GUI)

15 Introduzione ad Abstract Window Toolkit (AWT)

15 Struttura della libreria AWT ed esempi

15 Introduzione a Swing

15 Swing vs AWT

15 Le ultime novità per Swing

15 File JAR eseguibile

17 - Ciclo for migliorato ed enumerazioni

17 Ciclo for migliorato

17 Tipi Enumerazioni

18 - Varargs e static import