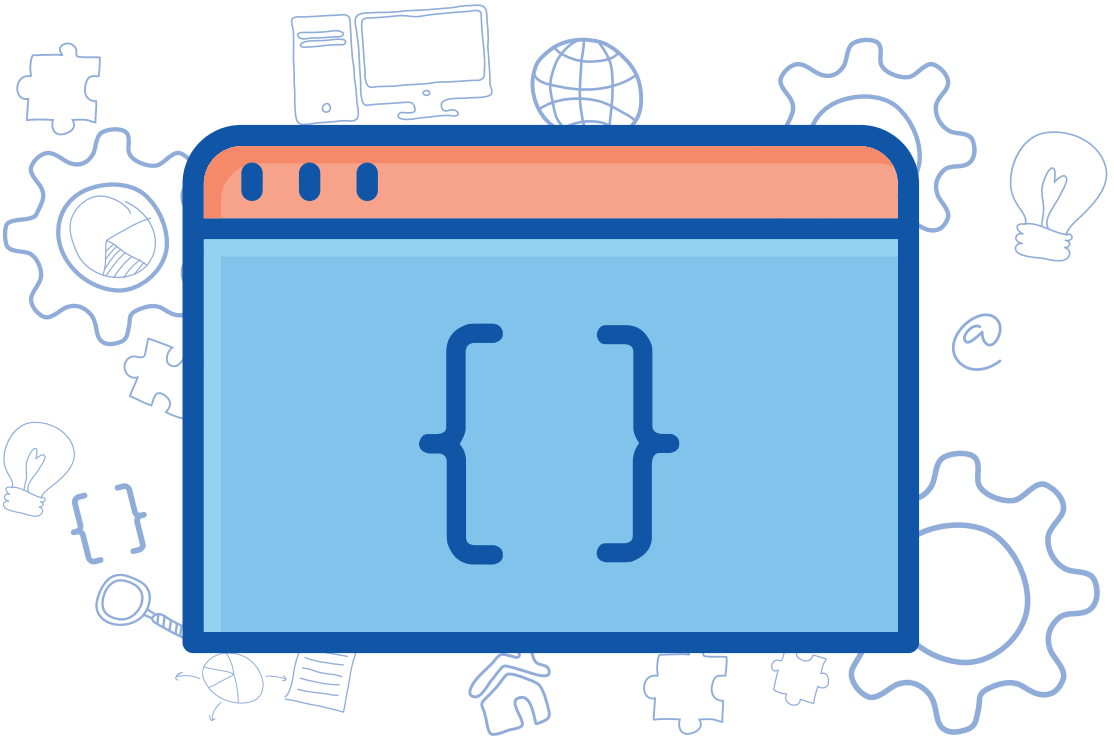
- Giuseppe Maggi -

PROGRAMMARECON

# Java 9

## Guida completa



**Variabili, tipi di dato, flusso di esecuzione, stringhe e array >>**

**La programmazione a oggetti, ereditarietà, eccezioni, generics >>**

**Strutture dati, espressioni lambda, database >>**

**JShell, i moduli e le altre novità di Java 9 >>**

*pro DigitalLifeStyle

*pro
Digital*LifeStyle

# Java 9

## Guida completa

**Giuseppe Maggi**

EDIZIONI
LSWR

Java 9 | Guida completa

**Autore**: Giuseppe Maggi

**Collana**: *pro DigitalLifeStyle

**Publisher**: Marco Aleotti
**Progetto grafico**: Roberta Venturieri
**Immagine di copertina**: © Alexey Blogoodf, © Vanatchanan | Shutterstock



**ISBN**: 978-88-6895-593-9



EDIZIONI LSWR

Via G. Spadolini, 7
20141 Milano (MI)
Tel. 02 881841
www.edizionilswr.it

Printed in Italy

Finito di stampare nel mese di febbraio 2018 presso "Rotolito" S.p.A., Seggiano di Pioltello (MI) Italy

(*) Edizioni Lswr è un marchio di La Tribuna Srl. La Tribuna Srl fa parte di LSWR GROUP.

# Sommario

1

# Iniziare a programmare in Java

**Lo scopo di questo capitolo è iniziare a programmare subito. L'impazienza è tanta ma dobbiamo attendere di predisporre l'ambiente necessario. Non c'è da preoccuparsi però: le operazioni saranno rapide e il nostro primo programma andrà in esecuzione prestissimo.**

## Brevissima introduzione: che cos'è Java e perché impararlo

Java è nato all'inizio degli anni Novanta, a opera della californiana Sun Microsystems, per offrire una piattaforma lavorativa adatta allo sviluppo di applicazioni efficienti, robuste e pronte a calcare i nuovi scenari connessi offerti dalla rete Internet. La missione è stata completamente assolta visto che tutte le principali statistiche vedono, ormai da anni, il linguaggio al primo posto nelle classifiche di utilizzo nel mondo, dopo aver superato anche C e C++, gli strumenti storici con cui quasi tutta l'informatica che oggi conosciamo è stata costruita.

Qual è stato il segreto di questo successo? Quali i punti di forza di Java?

Se dovessimo scegliere il suo elemento più rivoluzionario potremmo guardare all'**indipendenza dalla piattaforma** dei software prodotti. Ciò significa che un programma Java può essere scritto su un sistema operativo e utilizzato su qualunque altro. Prepariamo il nostro progetto su una macchina Windows perché questo è l'ambiente di lavoro che prediligiamo? Nessun problema: i nostri colleghi, committenti, clienti o chiunque ci abbia richiesto il software potranno utilizzarlo su una qualsiasi macchina Linux, Mac

o Windows a loro disposizione. Ciò mette in pratica uno dei motti preferiti degli ideatori di Java: **"Write once, run anywhere"** (tradotto: "Scrivi una volta, esegui ovunque"). I vantaggi di tutto ciò sono già immaginabili mentre il meccanismo che lo permette verrà indagato nei prossimi paragrafi.
I tratti caratteristici di Java sono però molti di più e qui proponiamo una rapida rassegna dei principali:

- **Java è una piattaforma** di lavoro, non solo un semplice linguaggio. Si tratta di un ambiente software, completo, ricco di strumenti già predisposti per supportare i nostri programmi. Proprio la ricchezza di risorse a disposizione di questo ambiente è qualcosa che stupisce chi vi si avvicina per la prima volta: si troveranno componenti già pronte per interagire con database, file e rete, gestire dati, eseguire calcoli, ottimizzare l'esecuzione e affrontare qualsiasi altra tematica sia necessario;
- tutti gli strumenti di lavoro indispensabili sono **gratuiti** sia per quanto riguarda la scrittura del software sia la piattaforma in se stessa. Non ci saranno costi obbligatori da pagare se non in termini di impegno, concentrazione e pazienza;
- Java è basato sulla **Programmazione Orientata agli Oggetti**, un approccio allo sviluppo che invita non solo a pianificare le azioni che il software dovrà compiere ma anche a progettare il modo in cui le informazioni saranno strutturate e collegate tra loro;
- la **sintassi** è sufficientemente elastica e intuitiva, simile a quella del linguaggio C (pertanto nota a molti sviluppatori già avviati) ma in una versione semplificata;
- **i software realizzati risultano robusti** anche grazie alle regole sulla gestione di tipi di dato cui il programmatore dovrà attenersi risolvendo così a priori potenziali fonti di *bug*;
- Java è seguito da una folta, volenterosa **comunità di programmatori**. Oltre a tutto il materiale di supporto che troveremo nella piattaforma, molte librerie software saranno rese disponibili – anche queste gratuitamente – da aziende, programmatori e fondazioni e le potremo integrare nel nostro progetto risparmiando molto lavoro.

Il prossimo paragrafo ci introdurrà all'ambiente di lavoro vero e proprio e ci accompagnerà alla realizzazione del nostro primo programma.

## Prepariamo l'ambiente di sviluppo

L'unico elemento imprescindibile per la programmazione in Java è il possesso di un kit di sviluppo, il **Java Development Kit (JDK)**. Viene distribuito sul sito ufficiale, gestito da Oracle (proprietario della tecnologia dal 2010) ed è disponibile all'installazione gratuitamente per qualunque sistema operativo all'indirizzo http://www.oracle.com/

technetwork/java/javase/downloads/index.html. Tra i pacchetti disponibili nelle stesse pagine, possiamo trovare anche il Java Runtime Environment (JRE) dedicato all'esecuzione degli applicativi: non avremo necessità di scaricarlo separatamente in quanto lo otterremo incluso nel JDK.

**L'installazione risulta molto semplice**. Una volta completato il download del pacchetto lo si potrà avviare seguendo le istruzioni per il proprio sistema operativo: comunque sia non si dovranno affrontare grosse difficoltà e si potrà lasciare l'impostazione di default per qualunque aspetto.

La piattaforma di lavoro sarà così posizionata nella nostra macchina e suddivisa in **due cartelle**: jdk-9 e jre-9. Tale suddivisione già rappresenta una rottura rispetto al passato in quanto fino alla versione 8 il Java Runtime Environment era contenuto in una sottocartella del JDK. Quest'ultimo è a sua volta articolato nella struttura visibile nella Figura 1.1.

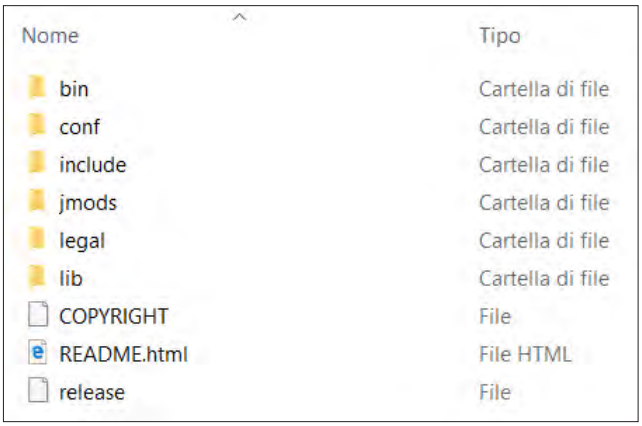



**Figura 1.1** – Struttura di cartelle del JDK.

Anche l'organizzazione interna del JDK è cambiata molto con la versione 9. In particolare, è apparsa la cartella *jmods* contenente i moduli – circa un centinaio di file .jmod – che costituiscono la piattaforma. La struttura modulare è la principale novità di Java 9 e sarà approfondita diffusamente nel seguito.

Per iniziare a programmare, la cartella *bin* riveste grande importanza. Al suo interno troveremo due comandi: **javac** e **java** con i quali si dovrà prendere confidenza al più presto. Il flusso di lavoro che porterà la nostra idea a diventare un programma Java fino a vederlo in esecuzione si potrà considerare suddiviso in tre fasi:

- **scrittura del codice sorgente**, quello in linguaggio Java per intenderci. Potremo usare un qualunque editor di testo come Notepad, gedit o qualsiasi altro avremo a disposizione. Il tutto verrà salvato in un file con estensione .java: parliamo al

momento di un solo file ma nel tempo ve ne saranno molti a costruire la struttura di un singolo progetto;
- **compilazione dei file .java**. Saranno trasformati da codice sorgente (formato leggibile e scrivibile dall'essere umano) in *bytecode*, pronto per essere eseguito come vedremo a breve;
- **esecuzione del bytecode** prodotto alla fase precedente. Contenuto in file di estensione .class, è destinato alla **Java Virtual Machine (JVM)**, altro componente della piattaforma.

I comandi *javac* e *java* servono ad attivare i meccanismi descritti agli ultimi due punti: rispettivamente, il compilatore e la Java Virtual Machine. In pratica, **con javac trasformeremo i nostri file .java in bytecode** (file con estensione .class) e **con java li manderemo in esecuzione**.
Tali *tool* dovranno essere invocati da riga di comando pertanto dovremo essere in grado di rintracciarli nel sistema operativo. In generale, si avranno due possibilità:
- li si potrà invocare premettendo l'indirizzo della cartella in cui risiedono. Immaginiamo per esempio di aver scelto Windows come nostro ambiente di lavoro e che il JDK sia stato collocato in C:\"Program Files"\Java\JDK; potremo invocare il comando *javac* come "C:\Program Files"\Java\JDK\bin\javac;
- si potrà impostare il PATH di sistema (il percorso in cui vengono automaticamente cercati i comandi invocati) in maniera tale che includa la cartella *bin* del JDK. Per ulteriori dettagli, si rimanda alla documentazione del proprio sistema operativo.

Prima di proseguire, sottolineiamo che il **cuore dell'indipendenza dalla piattaforma** dei programmi Java verte proprio sulla presenza della Java Virtual Machine. Il bytecode prodotto dalla compilazione non sarà espressamente destinato a un solo sistema operativo ma genericamente alla JVM. Quest'ultima è disponibile per qualsiasi dei principali sistemi operativi, pertanto la sua installazione trasformerà una macchina (qualunque sistema essa abbia) in un contesto ideale per il lancio in esecuzione del nostro programma.

## Le versioni di Java

Prima di iniziare a programmare, si vuole sottolineare ancora che Java negli anni è stato protagonista di una grande evoluzione e ogni tappa di questo percorso è stata segnata da una nuova versione. Di tale classificazione, si sentirà e si leggerà molto, non solo durante lo studio del linguaggio ma in tutto il tempo che se ne farà uso e ciò soprattutto riferito alla disponibilità delle nuove funzionalità via via introdotte.

La versione attuale è la 9 ma nel testo sottolineeremo spesso in quale aggiornamento del linguaggio una determinata caratteristica è stata aggiunta.
Non tutte le versioni hanno avuto la stessa importanza. Le più rivoluzionarie sono state la 5 e la 8: in questo libro infatti affronteremo capitoli interi dedicati alle loro innovazioni. Segnaleremo, in particolare, quali aspetti hanno visto la luce con Java 8 sia per la loro rilevanza sia perché si tratta di introduzioni recenti che pertanto non possono sempre trovare applicazione in progetti già esistenti, basati su versioni precedenti del linguaggio: non è raro incontrare scenari simili in ambito lavorativo soprattutto a breve tempo da rilasci di versioni molto significative.

NOTA

Quando inizieremo a utilizzare la documentazione ufficiale, noteremo che molte caratteristiche risulteranno introdotte nelle versioni 1.5, 1.6 o 1.8. Numeri simili si riferiscono alle versioni 5, 6 o 8 in quanto storicamente il numero con cui una versione del linguaggio Java diventava famosa era la prima cifra decimale.
Con la versione 9 del linguaggio questa abitudine è stata definitivamente interrotta infatti le nuove caratteristiche di Java 9 risulteranno introdotte a partire dalla versione 9 (nella documentazione troveremo la notazione "Since: 9").

## Il primo programma

Ogni corso di programmazione, indipendentemente dal linguaggio trattato, inizia con un classico esempio, comunemente chiamato "Hello world". Si tratta in genere di codice di base con cui viene stampata tale stringa, eventualmente tradotta in una lingua diversa dall'inglese.
Anche noi inizieremo da qui ma si badi bene che lo scopo dell'esercizio è sicuramente più articolato. Apparentemente, ci limitiamo a stampare questo semplice messaggio di saluto ma in realtà vogliamo:

1. illustrare il procedimento completo che porta dalla scrittura del programma in linguaggio Java fino alla sua esecuzione nella Java Virtual Machine;
2. predisporre la base di un elementare programma dal quale potremo partire per sperimentare ogni altro concetto che affronteremo insieme. In pratica, produrremo il nostro contesto per ogni futura esercitazione.

Iniziamo scrivendo il codice in un qualsiasi editor di testo:

```
public class Ciaomondo {
       public static void main(String[] args)
       {
         System.out.println("Ciao mondo!");
       }
}
```

e salviamo il file con il nome Ciaomondo.java. Possiamo sistemarlo in una cartella a nostro piacimento anche se l'ideale, soprattutto in fase di apprendimento del linguaggio, sarebbe raccogliere tutti i nostri esempi in un'unica collocazione.
Stiamo per iniziare ad analizzare ogni aspetto di questo primo esempio ma vogliamo premettere un paio di raccomandazioni:

- illustreremo il ruolo di ogni singolo elemento anche se molti concetti troveranno il giusto approfondimento nei capitoli successivi. Sarà nostra cura indicare quali elementi andranno presi "a fiducia" in questa fase in attesa di avere la preparazione sufficiente ad affrontarli;
- si tenga a mente sin da subito che **Java è un linguaggio case-sensitive** ossia considera differenti lettere maiuscole e minuscole. Il testo "Ciaomondo" per il nostro compilatore sarà del tutto diverso da "ciaoMonDo" o da "CIAOmondo". Porre la necessaria attenzione a questo aspetto eviterà errori che potrebbero farci perdere molto tempo.

Più che analizzare il testo riga per riga proveremo a smontarlo ricostruendolo dalle sue porzioni esterne verso quelle interne. Gli elementi che in questo esempio conosceremo della sintassi saranno di assoluta rilevanza in quanto costantemente ricorrenti in qualsiasi programma.

## Classi e blocchi di codice

Isoliamo la porzione esterna dell'esempio:

```
public class Ciaomondo {
      …
      …
}
```

Notiamo subito due aspetti: la parola chiave `class` e l'uso delle parentesi graffe.
La parola chiave `class` serve a definire una nuova classe. Si tratta di uno dei meccanismi cardine della Programmazione Orientata agli Oggetti ma qui, solo per il momento, ne facciamo un uso molto ridotto: lo impieghiamo per introdurre il blocco di codice che conterrà l'intero programma. La parola `class` è anticipata da `public`, questo è uno dei

modificatori di visibilità che in Programmazione a Oggetti permetteranno di stabilire le regole di utilizzo dei componenti inclusi nel progetto: per ora lo applichiamo sempre, il tempo del necessario approfondimento arriverà. **La classe pubblica che così andiamo a definire prende il nome di Ciaomondo**. Si noti che essa e il file che la contiene devono assolutamente chiamarsi nella stessa maniera, rispettando anche le differenze tra lettere maiuscole e minuscole.
**Le parentesi graffe delimitano un blocco di codice**. Intendiamo con tale espressione una o più righe in linguaggio Java (ma può anche essere vuoto) che costituiscono nell'insieme un'unità di esecuzione e sono asservite tipicamente a un costrutto. Questo blocco di codice delimita il contenuto della classe `Ciaomondo`.

## Il metodo main

Inseriamo ancora uno strato per avvicinarci all'esempio completo:

```
public class Ciaomondo {
       public static void main(String[] args)
       {
     …
     …
       }
}
```

Quello che appare all'interno della classe `Ciaomondo` (in pratica, nel blocco di codice a essa collegato) è il metodo `main()`. Il termine metodo rappresenta in Programmazione Orientata agli Oggetti il corrispondente di ciò che tipicamente in informatica viene detto funzione o procedura. Parliamo quindi di una o più righe di codice finalizzate al raggiungimento di uno scopo (esempio: realizzazione di un calcolo, elaborazione di alcuni dati, stampa di informazioni ecc.) ed etichettate con unico nome: grazie a questo nome il metodo potrà essere richiamato da un altro punto del programma.
La riga in cui viene definito il `main()`può avere adesso un'aria complicata:

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

Al momento non ce ne preoccuperemo, lo scriveremo così rimandando a più tardi approfondimenti sui singoli termini. Il metodo `main()` ha un ruolo speciale nel programma, ne **rappresenta l'entry point**. Quando lanceremo il nostro eseguibile frutto di compilazione, l'esecuzione inizierà proprio dal metodo `main()`. In questo esempio può apparire scontato essendo l'unico codice presente nel programma ma nei prossimi capitoli impareremo a realizzare progetti di più ampio respiro, composti da molti file e proprio tra uno di questi ci sarà la classe che offrirà il metodo `main`: da lì la Java Virtual Machine inizierà l'esecuzione.

Anche il metodo `main()` ha il suo blocco di codice, delimitato da una coppia di parentesi graffe. Si noti in merito che i blocchi di codice sono sempre contenuti l'uno nell'altro, non devono mai incrociarsi: ogni parentesi chiusa mette fine al blocco iniziato con l'ultima parentesi aperta.

### Il punto e il punto e virgola

In tutto l'esempio, l'unica riga davvero operativa è la seguente:

```
System.out.println("Ciao mondo!");
```

Qui appare uno dei protagonisti assoluti della programmazione Java: il **punto**. Lo troviamo, nel primo caso, tra i termini `System` e `out`. Il punto ci dice che l'elemento alla sua destra può essere trovato all'interno di quello nominato alla sua sinistra.
In pratica, con l'espressione `System.out` intendiamo che all'interno di `System` (un "contenitore" di strumenti utili per i nostri programmi Java) troviamo `out` che possiamo vedere come un canale di uscita di contenuti.
Analogamente, il seguente punto dice che `println()` può essere trovato all'interno di `out`. Ciò che chiamiamo `println()` è l'invocazione di un metodo e questo viene rivelato dalla coppia di parentesi tonde che lo seguono. Il loro scopo è racchiudere i dati con cui il metodo dovrà lavorare che in questo esempio consistono nella frase da stampare. Riepilogando (e semplificando), il metodo `println()` stampa i dati che gli passiamo mediante il canale `out` fornito tra gli strumenti di `System`. Ci dovremo abituare a questa sorta di "navigazione" tra gli elementi del codice e il punto è l'operatore di collegamento tra di essi.
Quello che viene fornito a `println()` è una stringa, un tipo di dato rappresentato da una qualsiasi sequenza alfanumerica: le stringhe avranno un capitolo a loro appositamente dedicato.
Al termine della riga, troviamo un **punto e virgola**. Questo è un altro elemento fondamentale che serve a indicare la fine di un'istruzione Java anche se ciò coinciderà generalmente con la fine della riga.

## Compilare ed eseguire il codice

Una volta preparato il primo programma di esempio, possiamo provarlo nella riga di comando del nostro sistema. Dovremo perciò cambiare la cartella corrente di lavoro (solitamente, nei vari sistemi operativi si può risolvere con il comando *cd*) spostandoci in quella in cui abbiamo salvato il file dell'esempio e invocare il comando *javac*, preceduto o meno dall'indirizzo completo a seconda se avremo cambiato il PATH di sistema. Supponiamo di utilizzare un sistema Windows e di aver inserito la cartella *bin* della nostra installazione del JDK nel PATH di sistema.

Il comando di compilazione sarà:

```
javac Ciaomondo.java
```

Se non si verificheranno errori, non otterremo alcun testo di risposta ma nella medesima cartella vedremo apparire il file Ciaomondo.class. Questo è il nostro bytecode e potremo eseguirlo così:

```
java Ciaomondo
```

e finalmente apparirà in output il messaggio "Ciao mondo!".
Si noti che nell'invocazione tramite comando *java* **non abbiamo dovuto specificare l'estensione .class** in quanto non stiamo invocando il file ma attivando la classe che possiede il metodo `main()`.

## Commenti

Capiterà molto spesso che vorremo inserire delle note all'interno dei nostri codici, degli appunti scritti in linguaggio naturale. I motivi di ciò potrebbero essere i più vari: documentare l'uso del programma, spiegare le ragioni di una scelta applicativa, fissare promemoria per i colleghi o per noi stessi in ottica futura e via dicendo. Insomma, per qualsiasi motivo sentiamo la necessità di lasciare tali annotazioni nel codice possiamo utilizzare dei **commenti**.
**Un commento può essere scritto su una o più righe** e possiamo usarne quanti ne vogliamo in una pagina di codice. Vediamo un esempio:

```
/*
     Commento su più righe:
     l'esempio stampa una sola stringa ma
     è un esercizio valido per iniziare a utilizzare Java
 */
public class Ciaomondo {
       public static void main(String[] args)
       {
         // Commento su una sola riga: il comando successivo stampa una stringa
         System.out.println("Ciao mondo!");
       }
}
```

Abbiamo mostrato l'utilizzo sia di un commento "multiriga" sia di uno a riga unica. Sintatticamente la differenza tra i due è che il primo, su più righe, va inserito tra le coppie di simboli `/*` e `*/` mentre il secondo tipo di commento viene segnalato dalla presenza di un doppio slash (`//`) a inizio riga.
Ripetendo le operazioni di compilazione ed esecuzione vedremo che **i commenti non influenzeranno minimamente le operazioni che il programma eseguirà** proprio per-

ché vengono esclusivamente considerati una disponibilità del programmatore ma non avranno alcun ruolo nel processo di costruzione dell'eseguibile.

## Utilizzare un IDE

Sinora si è parlato di scrivere il codice con un normalissimo editor di testo. Eppure a lungo andare il programmatore sente la necessità di utilizzare un ambiente di lavoro che sia in grado di supportarlo maggiormente e ciò per una serie di motivi:

- un normale editor di testo non fornisce indicazioni su errori di sintassi pertanto dovremo aspettare la compilazione per sapere se abbiamo sbagliato qualcosa;
- la piattaforma Java è molto ricca di componenti, ognuna con il suo nome e un ricco stuolo di proprietà e metodi da invocare: per quanto riguarda suggerimenti e completamento automatico del codice, un editor di testo non fornisce alcun supporto;
- si dovrà digitare ogni volta le direttive di compilazione ed esecuzione da riga di comando;
- il testing e il debug (ricerca di errori) dalla riga di comando risultano piuttosto difficoltosi.

Per questi e altri motivi, l'accoppiata editor di testo e riga di comando non è sufficiente al programmatore esperto di Java e quindi si dovrà ricorrere all'uso di un **IDE (Integrated Development Environment)**.
Si tratta di software visuali, piuttosto articolati, che mirano a offrire un sostegno totale al programmatore. Le funzioni primarie saranno: supporto all'editing del codice con suggerimenti, completamento del testo e controllo sintattico; esecuzione del programma passando, ovviamente, per la compilazione; debug per la ricerca di errori di funzionamento. Comunque le funzionalità offerte da un buon IDE non si limiteranno a queste ma saranno molte di più: questi software infatti saranno spesso basati su una struttura modulare che permetterà di integrare plugin per ottenere ulteriori capacità.
**Uno degli IDE più conosciuti del mondo Java è Eclipse**. Si tratta di un progetto open source, gratuito, multipiattaforma e modulare che offre nativamente supporto a Java ma dispone altresì di versioni per C/C++, PHP e altri linguaggi ancora.
Può essere scaricato dalla pagina dei "Downloads" del sito ufficiale http://www.eclipse.org e sarà necessario solo scegliere una versione compatibile con il proprio sistema operativo.
Ciò che si scaricherà sarà un *installer* che una volta avviato permetterà di scegliere quale versione di Eclipse installare. Per i nostri scopi andrà bene la prima versione proposta, "Eclipse IDE for Java Developers". L'*installer* si preoccuperà di scaricare tutto

il necessario e lo installerà in un percorso che potremo eventualmente personalizzare. Gli IDE – ed Eclipse non fa eccezione – lavorano a progetti piuttosto che a file singoli. Un **progetto** consiste in una struttura di cartelle in cui metteremo tutto il necessario per la creazione del programma: codice Java ovviamente ma anche risorse di altro genere come file di configurazione, dati, database e molto altro. In Eclipse, dovremo subito creare un progetto e questo sarà collocato in una cartella che l'IDE ci farà eleggere a **workspace** alla prima apertura.
Per sperimentare la classe `Ciaomondo` in Eclipse dovremo, prima di tutto, creare un nuovo progetto seguendo il menu *File > New > Java Project*. Non ci sarà da impostare nulla se non il nome che vorremo assegnargli.

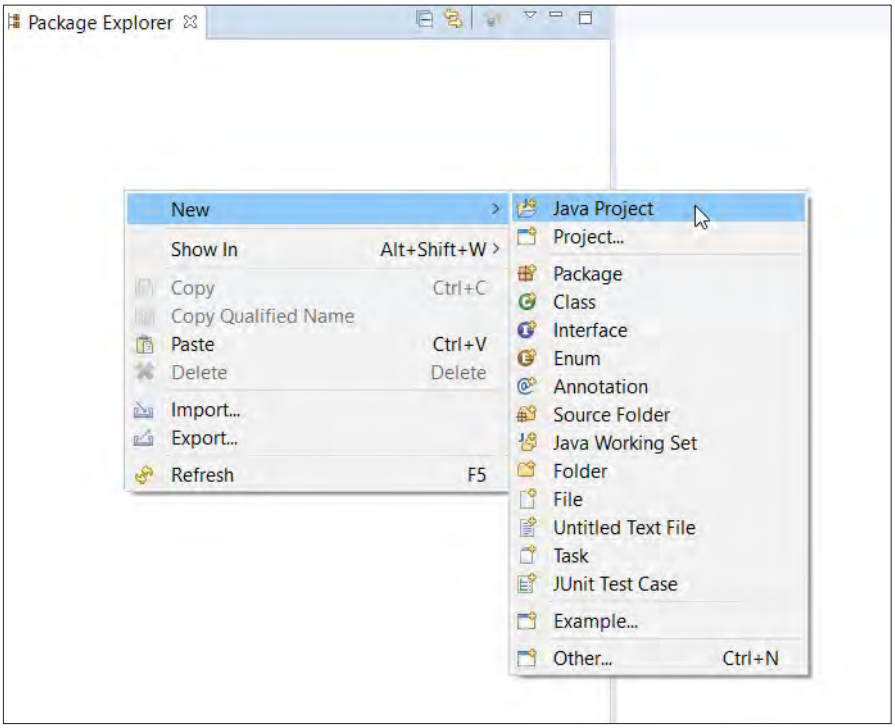



**Figura 1.2** – Creazione di un nuovo progetto in Eclipse.

All'interno del progetto, troveremo una cartella denominata *src* che sarà destinata a contenere i sorgenti in linguaggio Java. Qui dovremo creare una nuova classe alla quale assegneremo un nome e, se vorremo posizionare un `main()` al suo interno, attiveremo la checkbox che riporta l'etichetta "public static void main(String[] args)".

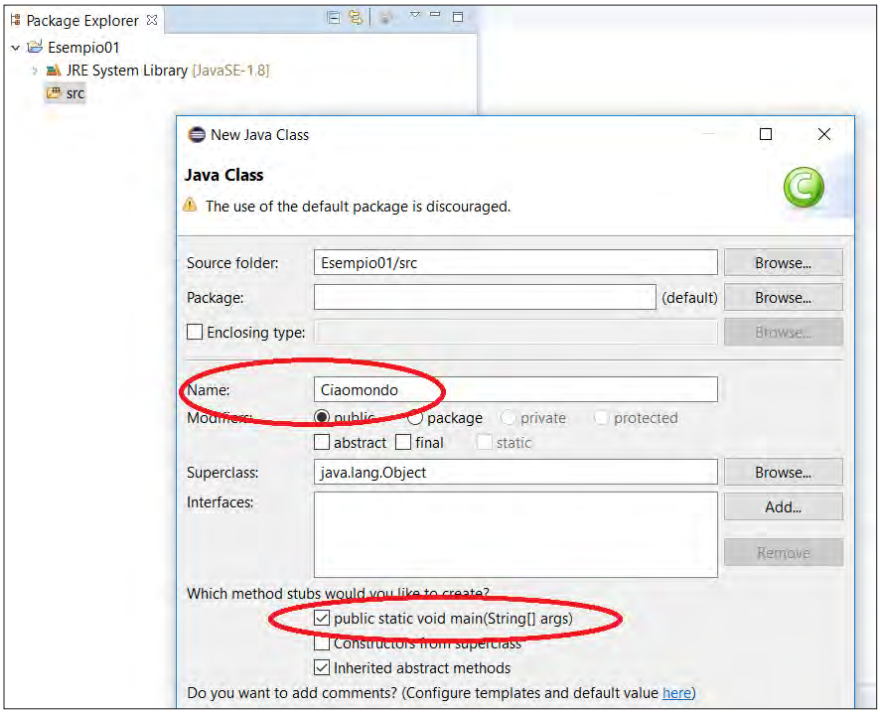



**Figura 1.3** – Creazione di una nuova classe Java.

All'interno del file che avremo aperto scriveremo il codice dell'esempio e avvieremo il tutto con il comando *Run* attivabile tramite il menu *Run* > *Run* o il pulsante presente nella barra a forma di cerchio verde con impresso un triangolo bianco.

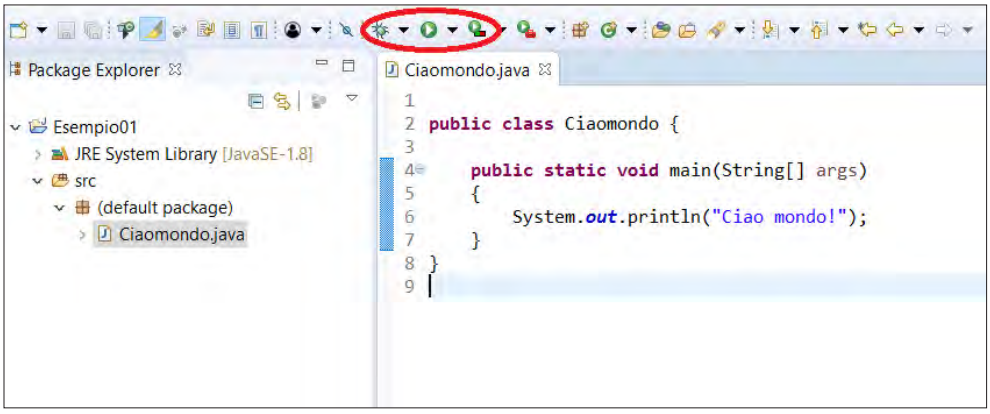



**Figura 1.4** – Scrivere codice in Eclipse.

Come risultato, vedremo apparire in basso una finestra che rappresenterà il nostro terminale che, se non ci saranno stati problemi, conterrà la stringa "Ciao mondo!".
Un uso approfondito di Eclipse richiederà un po' di pratica ma gli sforzi saranno ampiamente ripagati dall'utilità che troveremo in questo programma. Ultima nota: **un altro IDE molto conosciuto è NetBeans**, dal funzionamento molto simile ad Eclipse che potrà essere ottenuto dalla stessa pagina da cui abbiamo scaricato il JDK.

## Il codice usato nel libro

Tutti gli esempi di codice usati nel libro sono scaricabili al seguente indirizzo:
https://java9-guidacompleta.blogspot.it/2018/02/java-9-guida-completa.html

## Riepilogo

Per programmare in Java abbiamo bisogno di preparare gli strumenti necessari: tra questi, il **JDK** è il primo e indispensabile. Lo otterremo gratuitamente per qualsiasi sistema operativo direttamente dal sito dell'azienda Oracle. Per quanto riguarda la **scrittura del codice**, avremo due vie: utilizzo di un editor di testo o di un ambiente completo, un **IDE**, come il conosciutissimo Eclipse.
Successivamente, potremo scrivere il nostro primo programma rispettando i limiti dei **blocchi di codice** e inserendo un metodo `main()` all'interno di una classe pubblica (il cui nome dovrà coincidere con quello del file). Al termine di ciò, potremo compilare il codice sorgente (comando **javac**) e mandare in esecuzione il **bytecode** così prodotto (comando **java**). Utilizzando un IDE, questi ultimi passi potranno essere sintetizzati in un unico click sul comando *Run*.