Introduzione

Imparare un linguaggio di programmazione, e nello specifico Java, è come fare un viaggio in una terra lontana e sconosciuta. Dunque è, sì, un viaggio meraviglioso, affascinante e gratificante, ricco di sorprese e scoperte, ma anche un viaggio non semplice, che richiede tanta pazienza e il giusto tempo. In ogni caso, al termine del viaggio, la ricompensa sarà elevata: si sarà infatti appreso un linguaggio di programmazione, potente e flessibile, che permetterà di utilizzare API (*Application Programming Interface*) di qualsiasi tipo, da quelle più *tediose* per interfacciarsi a un database a quelle più *divertenti* per scrivere videogiochi, e dunque scrivere software di qualsiasi tipo.
Java, come avremo modo di verificare durante tutto il percorso di apprendimento che il libro intende offrire, è un linguaggio di programmazione estremamente espressivo e ricco di costrutti sintattici, e che dà grande libertà operativa al programmatore, il cui unico limite sarà solo la sua fantasia e la preparazione sulle regole sintattiche dei costrutti o sulla semantica delle operazioni.
Desidero spendere qualche parola anche sui principi ispiratori che mi hanno guidato nella scrittura del presente testo.

- Gli argomenti propri di ogni capitolo hanno un preciso e chiaro ordine. Ogni capitolo esprimerà compiutamente il relativo obiettivo didattico e non si sovrapporrà o comprenderà contenuti di altri capitoli (per esempio, se nel Capitolo 2 si parlerà delle variabili, solo nel Capitolo 3 si parlerà degli array). Quanto detto può apparire abbastanza ovvio ma non lo è. Infatti, oggi, si sta assistendo alla proliferazione di testi nei quali gli argomenti sono scritti "a spirale" dove cioè un argomento può contenere riferimenti iniziali ad altri argomenti, i quali saranno poi trattati approfonditamente solo nel capitolo di pertinenza. Questo, a mio parere, laddove non strettamente necessario e comunque non legato ai costrutti del linguaggio (è evidente che per mostrare il valore di una variabile bisogna dire qualcosa sul metodo `System.out.printf`) induce a distrazioni e fa perdere inutile tempo per la corretta comprensione della corrente unità didattica.
- Il modo espositivo che ho seguito è rigoroso, laddove necessario piuttosto formale, e ho prestato molta attenzione al corretto uso della terminologia propria di Java. Questo non è un libro del tipo *impariamo Java in 24 ore* oppure *Java for Dummies*. Java è un linguaggio complesso e ricco di costrutti; per insegnarlo ci vuole "serietà" e giusto rigore; per impararlo ci vuole pazienza e disciplina.

- Ho concepito i listati e gli snippet di codice in modo che dessero una chiara indicazione pratica dei relativi argomenti teorici; sono piuttosto brevi, autoconclusivi e decorati da commenti che danno, talvolta, ulteriori spiegazioni teoriche.
- Non ho trattato API specifiche della piattaforma Java ma solo quelle "connesse" con il linguaggio. Questo è un libro su Java, ossia sul linguaggio di programmazione, non un libro sulla programmazione in Windows, GNU/Linux o macOS. Ha senso spendere centinaia di pagine per parlare della programmazione di GUI, di database, del Web e così via, elencando di fatto una pletora di API? Ha senso sottrarre centinaia di pagine ai dettagli sui costrutti del linguaggio stesso per "donarle" a quelle API? Credo di no. Credo che un libro su Java *sia* un libro su Java, ossia un libro, per esempio, che spende centinaia di pagine per spiegare in modo compiuto, dettagliato e rigoroso che cos'è la OOP, la programmazione funzionale e così via per tutti gli altri argomenti a esso pertinenti. In definitiva, credo che un testo così strutturato dia al lettore, paziente e volenteroso, una marcia in più per poi affrontare con la giusta comprensione e consapevolezza le API che avrà interesse di apprendere e che, nel caso di Java, godono di un'abbondante documentazione.

Organizzazione del libro

Il libro è organizzato nelle seguenti parti.

- Parte I – *Concetti e costrutti fondamentali*, costituita dai seguenti capitoli: Capitolo 1, *Introduzione al linguaggio*; Capitolo 2, *Variabili, costanti, letterali e tipi*; Capitolo 3, *Array*; Capitolo 4, *Operatori*; Capitolo 5, *Istruzioni e strutture di controllo*; Capitolo 6, *Metodi*. Questa parte enuclea le nozioni essenziali e fondamentali che sono propedeutiche di un qualsiasi corso sulla programmazione: dal concetto di variabile e array, passando per gli operatori e le strutture di controllo del flusso di esecuzione del codice, finendo con un dettaglio su come scrivere blocchi di codice attraverso il costrutto di *metodo*.

  NOTA

  Nell'organizzazione dei capitoli ho preferito introdurre i metodi prima del costrutto di classe, poiché sono "strutture" sintattiche più semplici e in modo da affrontare i concetti essenziali del linguaggio in ordine crescente di complessità. Filosoficamente, questa scelta si sposa bene con l'inquadramento del paradigma a oggetti (basato sulle classi) inteso come estensione del paradigma procedurale (basato sulle procedure). Non a caso, le strutture di controllo della programmazione ad oggetti, che vengono poi utilizzate all'interno dei metodi, sono le stesse del paradigma procedurale.

- Parte II – *Paradigmi, stili di programmazione e gestione degli errori*, costituita dai seguenti capitoli: Capitolo 7, *Programmazione basata sugli oggetti*; Capitolo 8, *Programmazione orientata agli oggetti*; Capitolo 9, *Programmazione generica*; Capitolo 10, *Programmazione funzionale*; Capitolo 11, *Eccezioni e asserzioni*. Questa parte illustra come avvantaggiarsi, nella costruzione di sistemi software, dei più noti paradigmi di programmazione: da quello maggiormente utilizzato, proprio della OOP, a quello, definito funzionale, che oggi sta riscuotendo una profonda rivalutazione e un certo successo. Analizza anche come impiegare nei programmi uno stile di programmazione *generico* ossia che fa uso di tipi e funzionalità che sono in grado di compiere una medesima operazione su

più tipi di dato differenti. Spiega, infine, come intercettare e gestire adeguatamente gli errori software grazie all'utilizzo del meccanismo di gestione delle eccezioni.

- Parte III – *Concetti e costrutti supplementari e avanzati*, costituita dai seguenti capitoli: Capitolo 12, *Package*; Capitolo 13, *Moduli*; Capitolo 14, *Annotazioni*; Capitolo 15, *Documentazione del codice sorgente*. Questa parte dettaglia come organizzare, tramite i *package*, in modo raggruppato, strutturato e gerarchico, dei tipi che sono connessi a una particolare funzionalità applicativa; come decomporre un'applicazione in un insieme di *moduli* e utilizzare, dunque, nel suo insieme, l'importante *sistema a moduli* introdotto a partire dalla versione 9 del linguaggio; come associare, tramite *metadati* (annotazioni), informazioni descrittive agli elementi di un programma che decorano; come utilizzare tag "speciali" che consentono di formattare il codice sorgente in modo che sia possibile generare per esso un'adeguata documentazione.
- Parte IV – *Introduzione ai tipi e alle librerie essenziali*, costituita dai seguenti capitoli: Capitolo 16, *Caratteri, stringhe e blocchi di testo*; Capitolo 17, *Espressioni regolari*; Capitolo 18, *Collezioni*; Capitolo 19, *Programmazione concorrente*; Capitolo 20, *Input/Output: stream e file*. Questa parte analizza i caratterile, le stringhe e i blocchi di testo, le espressioni regolari, le collezioni di dati, la programmazione concorrente e le operazioni sui file. Un capitolo aggiuntivo sulla programmazione di rete è disponibile online all'indirizzo `https://bit.ly/apo-progjava`.

Struttura del libro e convenzioni

Gli argomenti del libro sono organizzati in capitoli. Ogni capitolo è numerato in ordine progressivo e denominato significativamente in base al suo obiettivo didattico (per esempio, Capitolo 2, *Variabili, costanti, letterali e tipi*).
I capitoli sono poi suddivisi in paragrafi di pertinenza, al cui interno possiamo avere blocchi di *testo* o di *grafica*, a supporto alla teoria, denominati in accordo con il seguente schema:

Tipologia NrCapitolo.NrProgressivo (NomeFileSorgente$_{opt}$*) Descrizione*$_{opt}$*.*

- *Tipologia* può indicare: un listato di codice sorgente (*Listato*), un frammento di codice sorgente (*Snippet*), la sintassi di un costrutto Java (*Sintassi*), un comando di shell (*Shell*), l'output di un listato o di uno snippet (*Output*), un'immagine (*Figura*) una tabella (*Tabella*), un decompilato di un file `.class` (*Decompilato*).
- *NrCapitolo.NrProgressivo* indica il numero di capitolo e il numero progressivo della relativa tipologia di blocco di testo o di grafica.
- *(NomeFileSorgente)* indica, opzionalmente, il nome del file di codice sorgente se il blocco di testo rappresenta un listato o uno snippet.
- *Descrizione* indica, opzionalmente, una descrizione del blocco di testo o di grafica.

Per esempio, il blocco "Listato 6.2 (ByValue.java)" indica il secondo listato di codice del Capitolo 6 avente come nome del file di codice sorgente ByValue.java e nessuna descrizione. Oppure il blocco "Snippet 3.12 (Snippet_3_12.java) Matrici e la proprietà length" indica il dodicesimo snippet di codice del Capitolo 3 avente come nome del file di codice sorgente "Snippet_3_12.java" e come descrizione "Matrici e la proprietà length".

Quanto ai listati, abbiamo adottato anche la seguente convenzione: i puntini di sospensione (…) eventualmente presenti indicano che in quel punto sono state omesse alcune parti di codice, presenti però nei file `.java` allegati al libro. Gli stessi caratteri possono talvolta trovarsi anche negli output di un programma eccessivamente lungo.

NOTA

I comandi di shell devono essere scritti *così come sono*, ossia senza inserire i caratteri di fine riga che invece sono presenti nel libro per garantire un'adeguata formattazione del testo.

Codice sorgente

Tutti i listati e gli snippet di codice sono disponibili all'indirizzo `https://bit.ly/apo-progjava` e su GitHub all'indirizzo: `https://github.com/thp1972/Libro_Java21`. Una volta raggiunta la pagina web indicata, possiamo scaricarli facendo clic sul pulsante *Code* e poi su *Download ZIP* (Figura I.1).

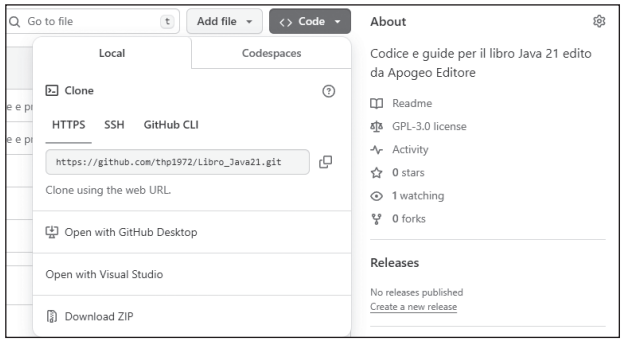

Figura I.1 Scaricare direttamente un repository tramite un archivio ZIP.

Alternativamente possiamo creare in locale (*cloning*) una copia del repository remoto del codice sorgente tramite un qualsiasi client Git.

DETTAGLIO

GitHub è, principalmente, una piattaforma web *cloud-based* dove è possibile archiviare, gestire e condividere progetti software grazie all'utilizzo di un sistema di controllo delle versioni (*version control system*) distribuito basato su Git, software sviluppato dallo stesso creatore di Linux, Linus Torvalds. Un sistema di controllo della versione è, in breve, un software che consente di tenere traccia nel tempo delle modifiche dei file di un progetto (per esempio, i file di codice sorgente), di ripristinarli a una determinata versione se qualcosa non funziona più, di far collaborare più persone su uno stesso progetto contemporaneamente, proteggere i file da perdite locali accidentali e così via. Infine, un *repository* è una sorta di cartella o archivio virtuale di un progetto che contiene file, codice sorgente, documentazione, immagini e, più in generale, tutti quegli asset che necessitano di un versioning.

Dopo aver ottenuto i file di codice sorgente avremo una struttura di directory così formata: `Libro_Java21` > `Codice` > `Capitolo_01`, `Capitolo_02` e così via. Avremo, cioè, una cartella per ciascun capitolo laddove queste ultime conterranno, a loro volta, le cartelle del codice sorgente Java divise per listati (`Listati`) e snippet di codice (`Snippets`).
Infine, mi preme dire che ho deciso di non vincolare il lettore dall'utilizzo di alcun IDE (*Integrated Development Environment*) in particolare, tipo NetBeans, Eclipse, Visual Studio Code e così via, preferendo invece adottare un approccio più "a basso livello" che si concretizza nell'insegnare come compilare ed eseguire codice Java direttamente a linea di comando con i suoi tool nativi: questo perché, data la natura multipiattaforma del linguaggio, il lettore può riutilizzare senza sforzo supplementare le nozioni apprese, e in modo coerente, su qualsiasi sistema operativo scelto sia esso Windows, GNU/Linux o macOS.

NOTA

All'indirizzo `https://github.com/thp1972/Libro_Java21` è disponibile il link `Installazione Java` che apre una guida per configurare e installare la piattaforma Java 21 sui sistemi Windows, GNU/Linux e macOS.

Il nuovo sistema di rilasci della piattaforma Java

Il 6 settembre 2017 un avvenimento importante ha scosso tutta la comunità Java ed è collegato a un articolo pubblicato sul blog di Mark Reinhold, *Chief Architect of the Java Platform Group* in Oracle, nel quale viene proposto, in breve, un cambiamento epocale nella temporizzazione dei rilasci della piattaforma Java e del JDK (*Java Development Kit*). Secondo Reinhold, infatti, l'ecosistema Java si è evoluto negli anni in maniera piuttosto irregolare e non ha mai avuto uno *schedule* programmato in modo temporalmente omogeneo. Si è data sempre maggiore priorità alle *big feature* integrabili nel linguaggio a scapito, magari, di piccoli anche se significativi cambiamenti. Ciò ha comportato che, finché queste feature non fossero completate e adeguatamente testate, la release finale della piattaforma slittasse di conseguenza.
Questi ritardi di rilascio, accettabili diversi anni fa quando piattaforme concorrenti si aggiornavano con la stessa lentezza, oggi non lo sono più perché le stesse piattaforme evolvono con una certa rapidità. Ecco dunque la necessità che anche l'ecosistema Java inizi a evolversi molto più rapidamente, al fine di garantirne un'adeguata competitività e cercando, comunque, di mitigare anche la normale tensione che esiste tra gli sviluppatori, ansiosi di vedere sempre più innovazioni e in tempi rapidi, e il mondo *enterprise* che invece desidera più stabilità e sicurezza.
Nelle parole di Reinhold: "Una cadenza di rilasci biennale, in retrospettiva, è semplicemente troppo lenta. Per raggiungere una cadenza regolare dobbiamo pubblicare versioni feature a ritmo più rapido. Posporre una feature da una versione a un'altra dovrebbe essere una decisione tattica dalle conseguenze minime più che una decisione strategica con impatto significativo. Per cui pubblichiamo una versione feature ogni sei mesi".
Ciò detto, vediamo dunque di esplicitare il predetto cambiamento sulle release di Java proposto da Reinhold e che è divenuto poi operativo a partire dal 20 marzo 2018 con il rilascio del JDK 10.
I rilasci di Java sono cambiati, nella sostanza, da un modello definito di tipo *feature-driven* (ogni rilascio era vincolato al completamento di importanti feature che poteva causare

anche diversi anni di ritardo) a un modello definito di tipo *time-driven* (anche detto *time-based*) che garantisce quanto segue.

- Una *feature release* ogni sei mesi che conterrà qualsiasi cosa: da API nuove o migliorate a feature proprie del linguaggio o della JVM (*Java Virtual Machine*). Ogni release dovrà essere disponibile a marzo e a settembre di ogni anno.
- Un *update* ogni tre mesi. Saranno limitati all'eliminazione di eventuali bug o alla risoluzione di problemi di sicurezza delle nuove feature. Ogni update dovrà essere disponibile a gennaio, aprile, luglio e ottobre di ogni anno.
- Una *long-term support release* (LTS) con una cadenza inizialmente triennale (JDK 11 a settembre 2018, JDK 17 a settembre 2021) e poi biennale (JDK 21 a settembre 2023, JDK 25 a settembre 2025, e così via). Le release con le date qui indicate sono quelle fornite direttamente da Oracle e avranno la garanzia di ricevere aggiornamenti in termini di performance, stabilità e sicurezza per tutta la durata del loro ciclo di vita.

Nella pratica questo nuovo modello di rilascio *time-based* potrà soddisfare sia gli sviluppatori, più inclini a volere in tempi rapidi innovazioni e nuove feature per il linguaggio (adotteranno le feature release con update semestrali ma dovranno sottostare ad aggiornamenti comunque semestrali: Java 10, Java 11 eccetera), sia il mondo enterprise, più incline, invece, a usare una release Java meno innovativa ma stabile (adotteranno una long-term support release con update garantiti per tutte le relative annualità e l'aggiorneranno almeno dopo tali periodi temporali).
Chiudiamo infine con una nota pratica su questo cambiamento dei rilasci di Java, che è esplicitato bene nel JEP 322: *Time-Based Release Versioning* e ha sostituito la possibile stringa di versione proposta sempre da Reihnold, la quale avrebbe dovuto avere il formato `$YEAR.$MONTH` (per esempio, per il 2018, la release di marzo sarebbe stata designata come 18.3, quella di settembre come 18.9 e così via).

> **TERMINOLOGIA**
>
> JEP è l'acronimo di *JDK Enhancement Proposals* e rappresenta un documento al cui interno sono inserite proposte di cambiamento, di miglioramento o di aggiunte di nuove feature alla piattaforma Java. All'indirizzo `https://openjdk.org/jeps/0` è presente una lista di tutti i JEP disponibili identificati da una denominazione e un numero progressivo.

In accordo, dunque, con il JEP 322, la stringa di versione per Java è attualmente costituita dai seguenti elementi:

- `$FEATURE`: incrementata ogni sei mesi. A marzo 2018 vale 10 (JDK 10), a settembre 2018 vale 11 (JDK 11) e così via.
- `$INTERIM`: varrà sempre 0 perché in questo modello di rilascio ogni sei mesi non si avrà mai una versione intermedia. Verrà comunque lasciata per motivi di flessibilità e possibilità di utilizzo.
- `$UPDATE`: verrà incrementata un mese dopo l'incremento di `$FEATURE` e poi ogni tre mesi. Ad aprile 2018 si ha il valore 1 (JDK 10.0.1), a luglio 2018 il valore 2 (JDK 10.0.2) e così via.

Riepilogando: da marzo 2018 è dunque partito un nuovo *release schedule* per la piattaforma Java che non prevederà più una big release con una moltitudine di cambiamenti ogni tre o quattro anni ma fornirà tante piccole release ogni sei mesi intervallate da update ogni 3 mesi e da una versione long-term support.

IMPORTANTE

Quando parliamo di long-term support ci riferiamo al supporto a pagamento fornito direttamente da Oracle per i suoi clienti e per la sua distribuzione del JDK (Oracle JDK). Comunque, da settembre del 2017, Oracle fornisce anche una release del JDK open-source e free sotto licenza *GNU General Public License, version 2, with the Classpath Exception* (OpenJDK). Oracle JDK è gratuito per lo sviluppo e il test, ma è a pagamento per l'uso in produzione. OpenJDK può essere utilizzato gratuitamente sia per lo sviluppo sia per la produzione. Infine, esistono anche altri vendors del JDK (RedHat, Amazon, Eclipse Foundation, e così via) che ne forniscono una propria distribuzione basata su OpenJDK e con un proprio programma di long-term support.

Da Java 10 a Java 21: tutte le modifiche del linguaggio

Al fine di mostrare l'enorme quantità di lavoro svolto dal team di ingegneri di Java a partire dalla nuova modalità di rilascio *time-based* delle nuove feature, mostriamo un elenco di tutte le modifiche apportate al linguaggio che possono servire anche come "bussola" per orientarsi velocemente su quale versione di Java contiene la feature ricercata o implementata.

- **Java 10, rilasciato il 20 marzo 2018.** *Local-Variable Type Inference* (JEP 286): estende la *type inference* (inferenza di tipo) alle dichiarazioni, tramite l'identificatore `var`, di variabili locali con inizializzatori.
- **Java 11, rilasciato il 25 settembre 2018.** *Local-Variable Syntax for Lambda Parameters* (JEP 323): consente di utilizzare l'identificatore `var` anche per la dichiarazione dei parametri formali di una *lambda expression*.
- **Java 12, rilasciato il 19 marzo 2019.** *Switch Expressions (Preview)* (JEP 325): estende la semantica della struttura di selezione multipla `switch` in modo che possa agire, oltre che come istruzione, anche come *espressione*, ossia come un costrutto che rappresenta o produce un valore.

TERMINOLOGIA

Una nuova feature del linguaggio, in una determinata release, può essere etichettata come *preview feature* (funzionalità di anteprima), ossia come feature la cui specifica, progettazione e implementazione è completa ma non permanente, ossia in futuro può subire cambiamenti, anche importanti, oppure essere addirittura eliminata da un JDK. Questo meccanismo (addirittura *codificato* in un apposito JEP, il 12) consente di ottenere prezioso feedback dalla vasta comunità di sviluppatori Java che possono in anteprima testare una specifica feature e dare indicazioni sui suoi punti di forza o di debolezza, segnalare se necessita di raffinamenti e miglioramenti, se ha errori tecnici, se manifesta scelte architetturali non ottimali e così via. Conseguentemente, la feature sottoposta a feedback può subire modifiche, ulteriori fasi di preview (spostata in altri JDK), essere infine integrata (viene eliminata l'etichetta preview) oppure venire rimossa ufficialmente dal JDK.

NOTA

Il presente libro, aggiornato alla versione 21 del linguaggio Java, si focalizzerà solo sulle feature che sono diventate effettive, non considerando quindi quelle con status di preview, a causa della loro possibile impermanenza o modifica nelle versioni future del linguaggio.

- **Java 13, rilasciato il 17 settembre 2019.** *Switch Expressions (Second Preview)* (JEP 354). *Text Blocks (Preview)* (JEP 355): aggiunge la possibilità di costruire letterali stringa su più linee con una sintassi flessibile e maggiormente leggibile.
- **Java 14, rilasciato il 17 marzo 2020.** *Pattern Matching for instanceof (Preview)* (JEP 305): introduce il *pattern matching* per l'operatore `instanceof` rendendo il codice più leggibile, conciso e sicuro. *Records (Preview)* (JEP 359): aggiunge la possibilità di dichiarare delle classi come di tipo *record*, ossia come dei tipi che conterranno, fondamentalmente, solo dati immutabili. *Switch Expressions* (JEP 361). *Text Blocks (Second Preview)* (JEP 368).
- **Java 15, rilasciato il 15 settembre 2020.** *Sealed Classes (Preview)* (JEP 360): aggiunge la possibilità di dichiarare classi o interfacce di tipo *sealed*, permettendo loro di specificare quali classi o interfacce possono estenderle o implementarle. *Pattern Matching for instanceof (Second Preview)* (JEP 375). *Records (Second Preview)* (JEP 384). *Text Blocks* (JEP 378).
- **Java 16, rilasciato il 16 marzo 2021.** *Sealed Classes (Second Preview)* (JEP 397).
  - *Pattern Matching for instanceof* (JEP 394). *Records* (JEP 395).
- **Java 17, rilasciato il 14 settembre 2021.** *Pattern Matching for switch (Preview)* (JEP 406): introduce il *pattern matching* per la struttura di selezione multipla `switch` rendendo il codice più leggibile, conciso e flessibile. *Sealed Classes* (JEP 409).
- **Java 18, rilasciato il 22 marzo 2022.** *Pattern Matching for switch (Second Preview)* (JEP 420).
- **Java 19, rilasciato il 20 settembre 2022.** *Pattern Matching for switch* (Third Preview) (JEP 427). *Record Patterns* (Preview) (JEP 405), consente di effettuare operazioni di *pattern matching* sui tipi *record* al fine, principalmente, di estrarne i dati in modo conciso ed elegante.
- **Java 20, rilasciato il 21 marzo 2023.** *Pattern Matching for switch* (Fourth Preview) (JEP 433). *Record Patterns* (Second Preview) (JEP 432).
- **Java 21, rilasciato il 19 settembre 2023.** *Pattern Matching for switch* (JEP 441). *Record Patterns* (JEP 440. *String Templates (Preview)* (JEP 430): introducono per i letterali stringa la possibilità di incorporare delle espressioni, accoppiate con processori di modelli, al fine di far produrre dei valori a *runtime* (*string interpolation*). *Unnamed Patterns and Variables (Preview)* (JEP 443): permette di specificare tramite il carattere trattino basso _ (*underscore*, valore Unicode `U+005F`) che il tipo e il nome (oppure solo il nome) di un componente di un record possano essere omessi dal *pattern matching*, oppure che l'identificatore che segue il tipo o `var` in un'istruzione o espressione sia altresì omesso. Questa funzionalità può essere utilizzata quando, per esempio, non si ha necessità di utilizzare in uno specifico contesto una determinata variabile (la si può ignorare) e questo al fine di semplificare il codice e migliorarne la leggibilità. *Unnamed Classes and Instance Main Methods (Preview)* (JEP 445): aggiunge la possibilità di creare classi senza nome (*unnamed classes*) e metodi `main` che siano anche di *istanza*. Queste feature consentono, soprattutto per i neofiti, di creare semplici e piccoli programmi con facilità e senza preoccuparsi da subito di imparare o approfondire concetti avanzati come il costrutto di classe, i package, i modificatori `static`, `public` e così via.