Java

Il codice è un file .java, in java il compiler prende il file e lo trasforma NON nel linguaggio macchina ma in byte code che ha estensione .class. il byte code è un linguaggio intermedio che sta a metà tra quello umano e quello della macchina, abbiamo bisogno della java virtual machine per far girare questo codice e una volta che la jvm run il codice viene convertito nel linguaggio macchina così che il pc lo possa comprendere. In C/C++ questo passaggio è saltato. Ma questo è il motivo per cui java è indipendente dalla piattaforma, si occupa la jvm a convertirlo nella macchina giusta.

Java è indipendente: il codice può girare su ogni sistema operativo. La java virtual machine è invece platform dipendente! Ogni sistema operativo ha una java virtual machine che si occupa di tradurre nel proprio linguaggio macchina il codice java.

Architettura di java

Jdk (java development kit) = jre + dev tools

È un pacchetto di file che fornisce l’ambiente per sviluppare e far girare il programma che creiamo con java. Comprende:

* Development tools
* Jre per eseguire il programma
* Il compiler – javac
* Archivio – jar
* Generatore di documenti – javadoc
* Interprete/loader

Jre (java runtime environment) = jvm + library class

Installa pacchetti che servono per eseguire il programma. Comprende:

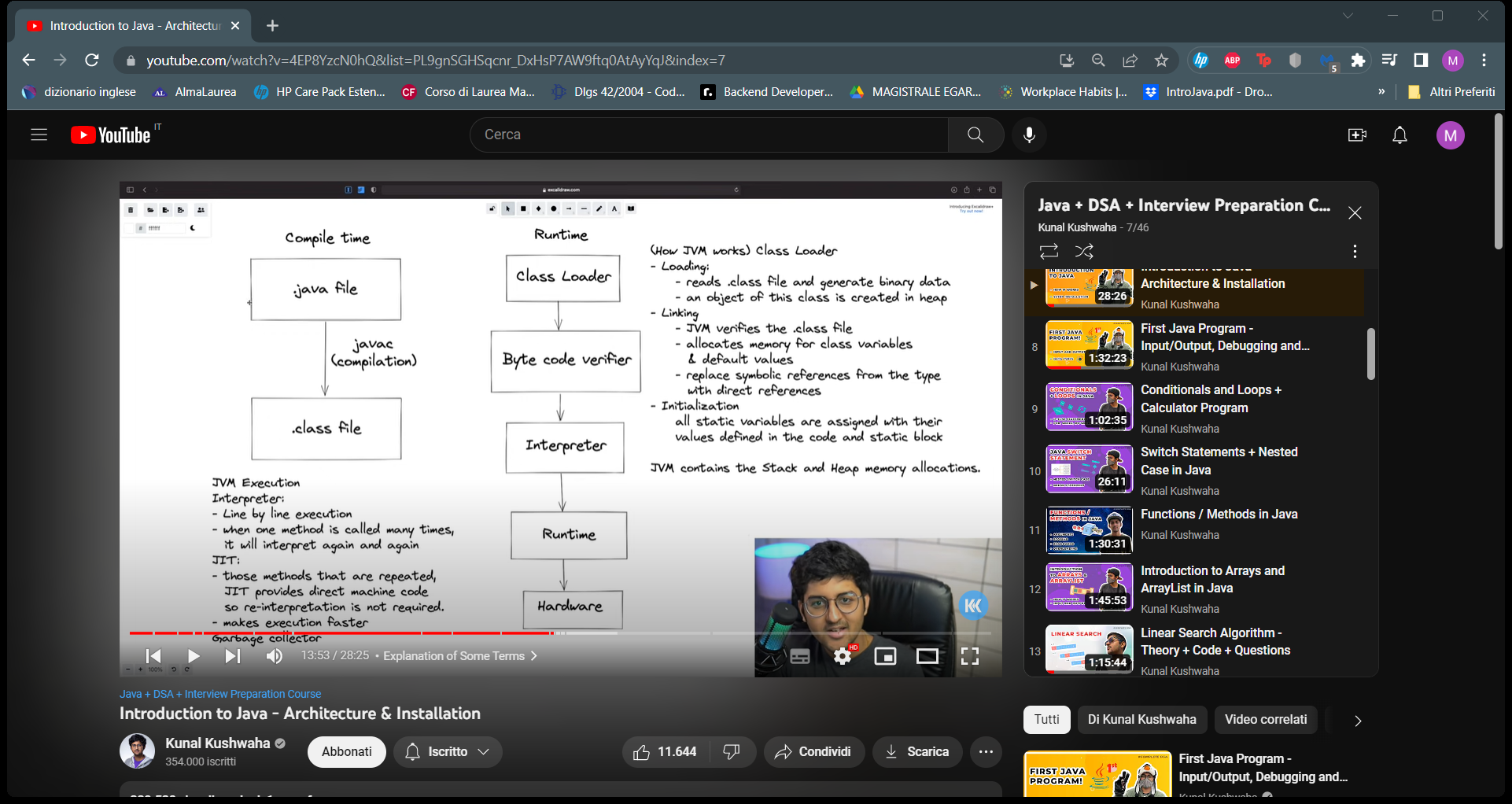
* Deployment technologie
* Interfaccia utente toolkits
* Librerie integrate
* Librerie di base
* Jvm

Dopo aver trasformato in .class quello che succede nella runtime è:

* Il caricatore delle classi carica tutte le classi che servono per eseguire il programma
* Jvm manda il codice al byte code verifier per controllarne il formato

Jvm (java virtual machine)

Jit (just-in-time)



public class Main {}

public: classe è accessibile a tutte le altre classi, file o package

Main è il nome della classe e deve essere sempre con la lettera maiuscola, deve avere il nome del file

Funzioni: blocchi di codice che possiamo riutilizzare all’infinito

Come funziona il compiler?

Scriviamo un semplice programma chiamandolo Main.java:

public class Main{

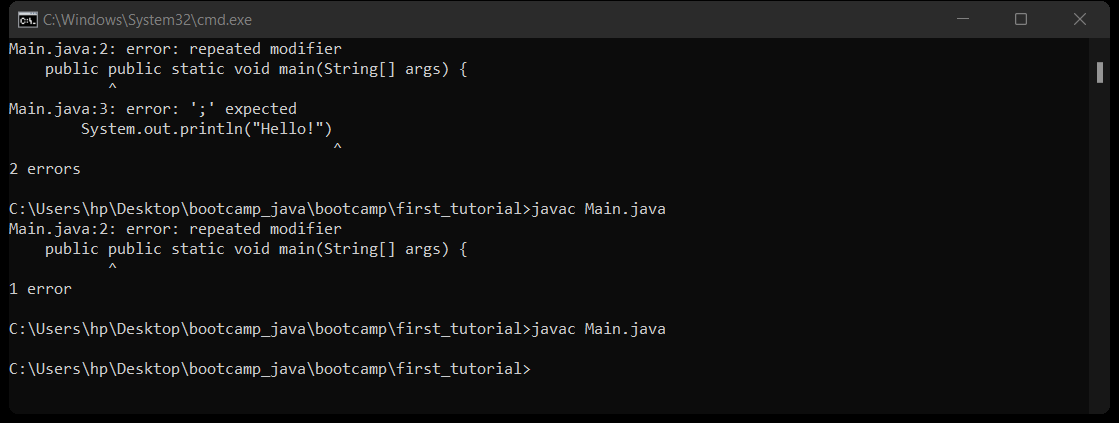
public static void main(String[] args) {

System.out.println(“Hello”)

}

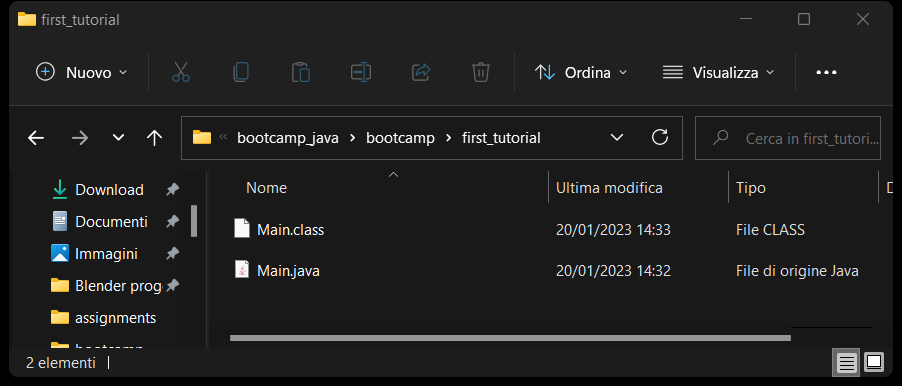
}

Per farlo compilare, trasformarlo in byte code scriviamo sul terminale javac + nome del file.java all’interno del nostro terminale.



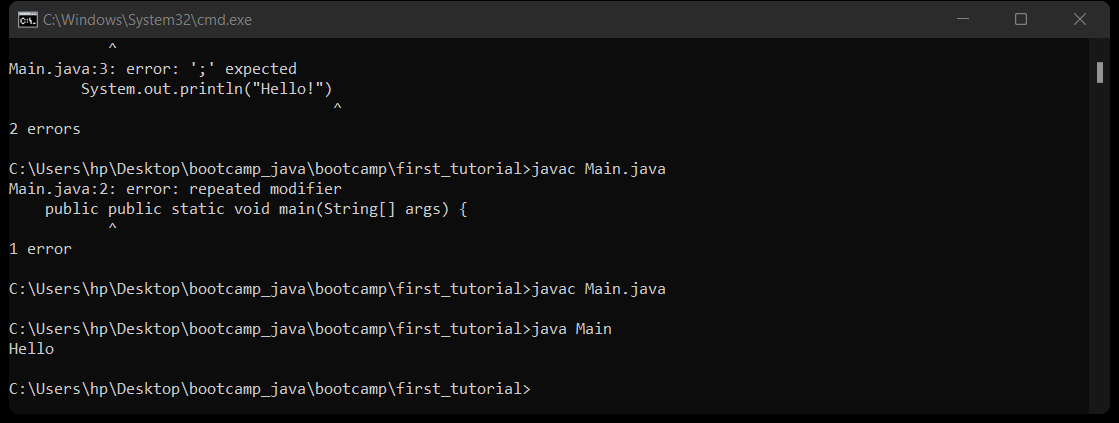


Dopo che avrà compilato correttamente comparirà nella cartella un file.class, quello sarà il nostro byte code. Possiamo scegliere dove salvare il byte code scrivendo javac -d .. + nome file. .. indica la directory prima, quindi nel nostro caso la cartella bootcamp. Di solito il byte code non va insieme al source code ma in folder diversi.





Per eseguirlo basta che scriviamo nel terminale java + nome





Public class Main:

Public: classe accessibile da ovunque

Classe: insieme di metodi e proprietà

Il nome del file DEVE ESSERE uguale a quello della CLASSE!

Public static void main (String[] args):

Il main: deve esserci in ogni classe.

Public: perché non avrebbe senso non rendere questo metodo visibile a tutti.

Static: il metodo main appartiene alla classe Main, il metodo fa riferimento alla classe in cui sta. Ci permette di invocare la classe senza creare un oggetto Main.

Void: return type, non vogliamo che ci torni nessun valore.

(String[] args): collezione dell’argomento, è un array di stringhe

Nel nostro progetto java abbiamo inserito il nostro file in un package e scritto all’interno del file qual è il package. Se vogliamo scrivere delle regole valide solo per il package in cui abbiamo messo alcuni file. Es: un metodo può essere acceduto solo ai file presenti in un determinato package. Questo ci aiuta con una maggiore sicurezza

Come funzionano gli output in java?

System.out.println();

La classe System è contenuta nel package lang, un package creato dai creatori di java che contiene una serie di metodi e classi per aiutarci a fare cose diverse.

Out: la classe system ha una variabile di nome out che è un tipo di PrintStream (da la capacità ad un output di stampare vari tipi di dati). Di default l’output stream è la command line ma possiamo stampare quello che vogliamo non sul terminale ma su un altro file e salvarlo da qualche parte specificando out = file prima di scrivere il syso.

Println() è un metodo di PrintStream

Come funzionano gli input in java?

Utilizziamo la classe Scanner che è all’interno del package java.util quindi dovremmo importare java.util.Scanner; per poterla usare.

Se Andiamo a vedere possiamo usare lo Scanner nella nostra classe perché è public!

Scanner è un semplice scanner di testo che può analizzare primitivi e reference usando espressioni regolari.

Per poterlo usare dobbiamo specificare che deve essere in grado di prendere input e dobbiamo quindi creare anche un nuovo oggetto:

Scanner input = new Scanner()

All’interno delle parentesi potremmo metterci qualsiasi cosa, potrebbe essere un file ad esempio. Nel nostro caso vogliamo che prenda l’input dalla tastiera quindi usiamo System.in;

Scanner input = new Scanner(System.in);

Possiamo notare che per stampare abbiamo usato System.out (il default output stream) mentre per leggere l’imput usiamo System.in (il default input: la teastiera)

Alla fine del codice dobbiamo chiudere lo scanner, per farlo usiamo il nome della variabile utilizzata per l’oggetto .close();

Primitivi

Cosa sono? Sono tipi di dati che non possono essere spezzati in ulteriori tipi di dati. Es:

int num = 59; non possiamo romperlo in altri tipi di dati

String name = “Miriam” possiamo spezzarlo in singoli caratteri char ‘M’ ‘i’ ‘r’ ‘i’ ‘a’ ‘m’.

Char let = ‘a’; non può essere spezzato in ulteriori tipi di dati

Float marks = 98.5f (PER I FLOAT E PER I LONG (numeri giganti integer) VANNO SEMPRE MESSE LE LETTERE L E F FINALI)

Il motivo per cui aggiungiamo f e L è per la dimensione degli integer, dei long, dei float e dei double.

Integer e float = 4 byte

Long e double = 8 byte

Solitamente si usano i double proprio perché ci permettono di conservare numeri più grandi. Per scrivere i numeri grandi separando le decine, migliaia ecc usare \_ es:

234\_000\_000

Anche questi tipi di dati contengono delle classi. Le wrapped integer, ad esempio, al posto di int. Questo si fa per avere delle funzionalità maggiori

Type Casting

public class TypeCasting {

public static void main(String[] argse) {

Scanner input = new Scanner(System.***in***);

float num = input.nextFloat(); //se qui al posto di un float metto un int java me lo casta automaticamente in un float

System.***out***.println(num);

input.close();

}

Nell’esempio qui sopra java fa un casting automatico (widening), mette l’int dentro un float. Il casting automatico avviene se si verificano diverse condizioni:

1. I due tipi devono essere compatibili! Non può essere un int e una stringa
2. Il tipo di destinazione deve essere più grande del tipo della sorgente, tutto ciò che è a sinistra dell’uguale deve essere più grande di ciò che è a destra!

Possiamo fare una conversione al contrario solamente specificando (narrowing)

Regole della promozione: TUTTE le operazioni se fatte tra tipi diversi vengono convertite nel tipo più grande es:

System.out.println(3 \* 5.634567654f);

risultato: 16.903704

Trasforma in automatico il 3 in un float!