IL LINGUAGGIO DI PROGRAMMAZIONE JAVA

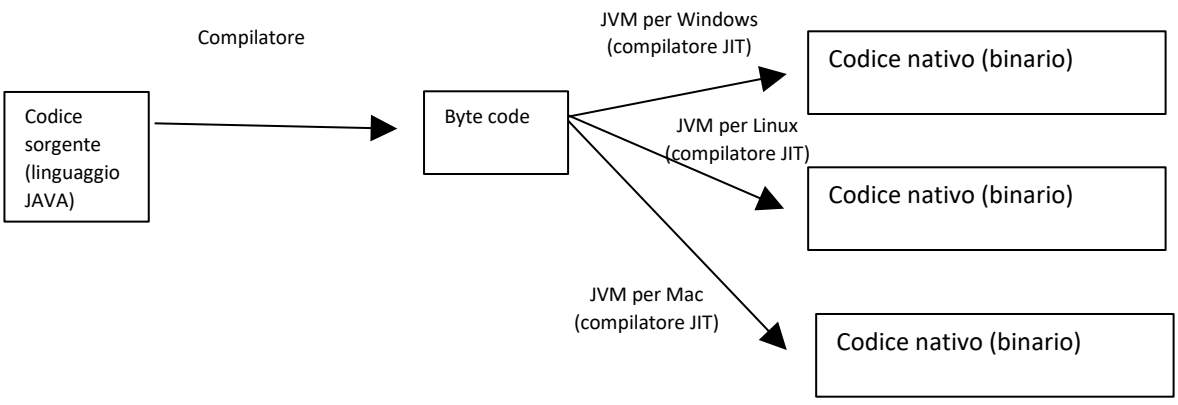
1. Soluzione Java fra compilatore e interprete

Come già visto l'anno scorso, Java ha ideato una soluzione per la traduzione in binario del codice sorgente, soluzione che rende estremamente portabili i software scritti con questo linguaggio di programmazione. Questa "soluzione java" è una "via di mezzo" fra l'approccio compilato e quello interpretato. Il compilatore java non genera un **codice binario** specifico per una determinata piattaforma HW/SW ma un codice "intermedio" chiamato **byte-code**. Il byte-code viene interpretato a runtime (ossia "durante l'esecuzione") da una "macchina virtuale" chiamata **Java Virtual Machine** (**JVM**). Per ogni piattaforma SW (ossia per ogni sistema operativo) esiste una JVM. La JVM va installata sul PC poichè essa è il software che interpreta il byte-code. La JVM rappresenta quindi una **astrazione** di un computer per qualunque piattaforma HW/SW. Il vantaggio di questo approccio è che, quando il programmatore scrive del codice in Java e poi genera il byte code, questo bytecode sarà eseguibile da ciascun computer reale, indipendentemente dal sistema operativo installato, a condizione che la macchina abbia installato la JVM specifica per quel sistema operativo. Per questo motivo il motto di Java è: ***"write once, run everywhere".***

Quando con un doppio click si manda in esecuzione un programma scritto in java, ciò che accade è che il byte-code viene convertito in codice binario dalla JVM. Questa traduzione non coinvolge tutto il bytecode ma riguarda solamente alcune parti, ossia le "parti" che vengono effettivamente eseguite dal processore, e tale traduzione avviene solamente nel momento in cui le funzioni vengono invocate (per questo motivo la traduzione in binario del bytecode è chiamata anche compilazione **Just In Time**, **JIT**, appena in tempo) analogamente a quanto avviene con un software interpretato. Le funzioni invocate, dopo esser state tradotte in binario ed eseguite, rimangono nella memoria centrale per eventuali successive invocazioni durante l'esecuzione, ma se alcune funzioni non vengono mai invocate durante l'esecuzione, esse non vengono mai tradotte da bytecode in binario. Questa modalità di traduzione consente di non dover tradurre in binario parti di codice non necessarie, velocizzando l'esecuzione. La velocità di esecuzione comunque non potrà mai raggiungere i livelli prestazionali di un programma eseguibile compilato **nativamente** (ossia "direttamente in binario" ad esempio in C) a causa dell'ulteriore livello di astrazione introdotto dalla JVM.

L'approccio JAVA è considerato una via di mezzo fra l'approccio compilato e quello interpretato perché:

1. Il codice sorgente viene tradotto in byte code (come nel compilato) ma ciò che si ottiene dalla traduzione non è il codice binario eseguito (a differenza dell'approccio compilato).
2. Il byte code viene tradotto in binario a runtime (come nell'approccio interpretato) ma una sola volta (a differenza dell'approccio interpretato in cui la traduzione istruzione per istruzione avviene ogni volta che l'istruzione viene eseguita dall' esecutore)



Quando nasce il linguaggio Java e perché ha avuto tanto successo?

Il linguaggio Java nasce nella prima metà degli anni '90 del secolo scorso, da un progetto della SUN Mycrosystem (poi acquisita nel 2009 dalla società Oracle) ad opera di un gruppo di lavoro guidato da **James Gosling**, ed è un linguaggio orientato agli oggetti.

Nasce come linguaggio di programmazione **orientato agli oggetti** per dispositivi embedded (ossia processori integrati a bordo di macchinari, ad esempio elettrodomestici, caldaie, automobili ecc.), con una sintassi simile al C/C++ (per facilitarne la diffusione fra i programmatori) ma semplificando alcuni meccanismi che risultavano di non semplice gestione in C++, ad esempio l'uso dei puntatori.

Java deve il proprio successo al fatto che nel 1995, nel periodo di grande incremento di utilizzo della rete internet, il browser Netscape (all'epoca il più diffuso), integrò la tecnologia per eseguire su pc in locale degli applicativi il cui codice era inviato da remoto, le **applet java.** Fino ad allora, le pagine web erano realizzate solamente in html e quindi erano statiche, ossia costituite solamente da testi e link. Per inserire nelle pagine web delle animazioni che le rendessero dinamiche, si trovò la soluzione di inviare, insieme al codice HTML, del codice che potesse poi essere eseguito sul pc client.
Il ragionamento era il seguente: sappiamo che ogni piattaforma hardware/software può eseguire del codice binario solamente se questo è stato compilato per tale specifica piattaforma. Quando un web server invia una pagina web ad un client, per poter inviare anche del codice eseguibile, sarebbe necessario che il web server riconoscesse la piattaforma hw/sw del client che ha richiesto la pagine, e che inviasse del codice binario diverso in base alla piattaforma installata sul client. Questa soluzione è molto complessa e di difficile manutenzione poiché renderebbe necessario modificare i file binari inviati insieme alle pagine web ogni volta che avvenisse una qualsiasi modifica su un qualsiasi sistema operativo. Si scelse allora di dotare i client di un ambiente di esecuzione (la JVM) che, una volta installato, fosse in grado di tradurre nel codice binario opportuno il bytecode inviato attraverso la rete insieme alla pagina web. Questo approccio, oltre ad aumentare estremamente la **portabilità** di un codice (*write once, run everywere*), garantiva un certo livello di **sicurezza** perché la JVM poteva, oltre a tradurre il codice binario, rilevare eventuale codice maligno (ad esempio rilevando istruzioni per la creazione e cancellazione di file) ed eseguirlo solamente se autorizzate dall'utente. I browser più recenti (Chrome e Edge) non supportano più le applet java, ma l'idea di installare la JVM sul PC client si è dimostrata vincente nella realizzazione di applicazioni desktop, tanto da essere poi utilizzata anche da Microsoft nel proprio linguaggio di programmazione ad oggetti C#). L'equivalente della **JVM** di Microsoft si chiama **CLR** (Common Language Runtime).

Quindi, riassumendo, gli elementi che hanno favorito la diffusione del linguaggio java sono:

1. **il fatto che è nato per la OOP e quindi dispone nativamente di numerose classi "già pronte" realizzate per determinate operazioni, ad esempio per la comunicazione in rete.**

2. **Portabilità (indipendenza dalla piattaforma).**

3. **Sicurezza**

2. Cosa sono JRE (Java Runtime Enviroment) e JDK (Java Developement Kit)

(Link al video su questa lezione: https://www.youtube.com/watch?v=odOxIdSguac&list=PLNrWrNHrd0qHhtrcR7KhBW_MPhNeX6u9&index=2)

Che cosa ci serve per eseguire un programma JAVA?

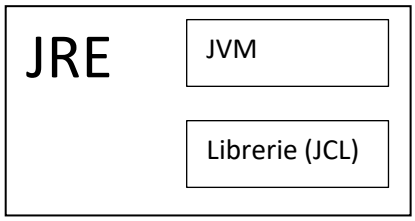
Per **l'esecuzione** di un programma java (ossia di codice in bytecode) è necessario che sia installato sul pc il **Java Runtime Enviroment** (**JRE**, ambiente di esecuzione java), esso comprende:

- la JVM
- JCL (Java Class Library): alcune librerie di classi caricabili dinamicamente che possono essere richiamate a runtime.

Se il nostro pc in passato ha già eseguito dei programmi in java, il JRE è già installato. Il JRE è ciò che installiamo quando ci viene chiesto, per eseguire un software, di "scaricare JAVA".
Per verificare se il JRE è già installato sul nostro PC:

menu cerca/configura Java → si apre il pannello di java, nella scheda java ci sono le informazioni del JRE

La versione più recente del JRE nel 2023 è la versione 8.
Se sul nostro PC non è installato il JRE non è un problema perché tale piattaforma è inclusa nel JDK, che ci serve per **programmare** in java, e che installeremo dopo.



Che cosa ci serve per scrivere programmi in java?

Ci serve un compilatore java, ossia il software che traduce il codice sorgente, in bytecode. In realtà, oltre al compilatore si scarica dal sito della Oracle un **JDK** (Java Developement Kit), che contiene diversi software di utilità per la programmazione:

- Compilatore (**Javac**)
- **Javadoc**: software per la documentazione del codice (vedremo in TPS)
- **Jar**: Java archivier, software che prepara un file compresso contenente una serie di librerie fra loro collegate

Qual è la differenza tra JRE e JDK ?

JRE **(Java Runtime environment)**	**JDK** **(Java Development Kit)**
È un'implementazione della Macchina virtuale Java* che esegue effettivamente i programmi Java.	È un insieme di software che potete utilizzare per sviluppare le applicazioni basate su Java.
Java Runtime Environment è un plug-in necessario per l'esecuzione dei programmi Java.	Java Development Kit è necessario per sviluppare nuove applicazioni Java.
JRE è più piccolo di JDK, quindi necessita di meno spazio sul disco.	JDK richiede più spazio sul disco poiché contiene JRE tra i diversi strumenti di sviluppo.
JRE può essere scaricato/supportato gratuitamente da java.com	JDK può essere scaricato/supportato gratuitamente da oracle.com/technetwork/java/javase/downloads/
Include JVM, le librerie di base e altri componenti aggiuntivi per eseguire le applicazioni e le applet scritte in Java.	Include JRE, l'insieme delle classi API, il compilatore Java, Web Start e i file aggiuntivi necessari per scrivere le applet e le applicazioni Java.

Approfondimento: quale è la società che fornisce l'ambiente di sviluppo JDK?
Come detto, il progetto Java è stato inizialmente sviluppato dalla SUN Mycrosystem. Tale società è stata poi acquisita dalla società Oracle. Quindi Oracle distribuisce il JDK. La versione che utilizziamo noi (chiamata JDK SE che sta per Standard Edition) è distribuita gratuitamente con licenza open source. La versione "professionale" utilizzata per sviluppare grandi applicazioni è chiamata JDK EE (Enterprise Edition). Di questa versione esistono sia release distribuite con licenza open source sia release con particolari funzionalità che vengono distribuite a con licenza proprietaria, quindi a pagamento. Link al download di Oracle: https://www.oracle.com/it/java/technologies/downloads/

Oltre ad Oracle esiste un'altra società che distribuisce con licenza open source sia il JDK SE sia il JDK EE. Tale società si chiama RedHat e la sua versione del JDK si chiama OpenJDK.
Il JDK di Oracle e OpenJDK vengono sviluppati parallelamente e non hanno differenze significative nel loro utilizzo. Link al download delle versioni di OpenJDK: https://developers.redhat.com/products/openjdk/download#assembly-field-downloads-page-content-82031
Esistono poi altre società che rilasciano le proprie distribuzioni dei JDK basate su OpenJDK.

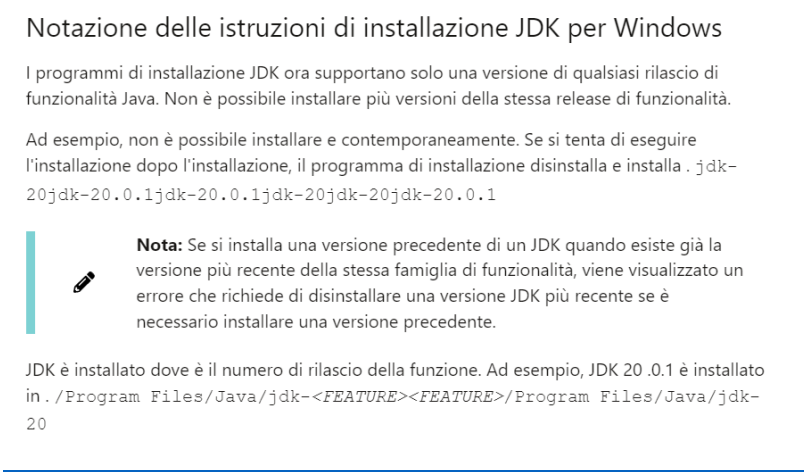
Citazione dal sito di RedHat:
"La più grande differenza tra OpenJDK e Oracle JDK è che OpenJDK è un progetto open source gestito da **Oracle, da Red Hat e dalla comunità**, mentre Oracle JDK è closed source, richiede una licenza a pagamento ed è gestito da **Oracle**. Con questa differenza, ci sono alcune funzionalità che non sono disponibili con OpenJDK poiché le funzionalità sono closed source o limitate dalla licenza."
https://www.redhat.com/en/topics/application-modernization/openjdk-vs-oracle-jdk#:~:text=The%20biggest%20difference%20between%20OpenJDK,and%20is%20maintained%20by%20Oracle.

Le versioni del JDK di Oracle sono aggiornate ogni 6 mesi. Alcune versioni sono più importanti di altre poiché per tali versioni viene garantito il supporto (aggiornamenti, correzione bachi ecc.) a lungo tempo.

Scarichiamo ed installiamo il **JDK 21** rilasciata il 19 settembre 2023. Questa sarà una versione LTS (Long Time Support) per la quale sono garantiti il supporto e gli aggiornamenti fino al 2031. Quando generalmente si parla di "**versione del linguaggio JAVA installato" si intende la versione del JDK**. Pertanto ora installiamo la "versione 21 di JAVA". Scegliamo la versione adatta al nostro OS dal sito di Oracle che si trova al seguente link:

https://www.oracle.com/java/technologies/javase-downloads.html

Approfondimento:



Notazione delle istruzioni di installazione JDK per Windows

I programmi di installazione JDK ora supportano solo una versione di qualsiasi rilascio di funzionalità Java. Non è possibile installare più versioni della stessa release di funzionalità.

Ad esempio, non è possibile installare e contemporaneamente. Se si tenta di eseguire l'installazione dopo l'installazione, il programma di installazione disinstalla e installa . `jdk-20jdk-20.0.1jdk-20.0.1jdk-20jdk-20jdk-20.0.1`

Nota: Se si installa una versione precedente di un JDK quando esiste già la versione più recente della stessa famiglia di funzionalità, viene visualizzato un errore che richiede di disinstallare una versione JDK più recente se è necessario installare una versione precedente.

JDK è installato dove è il numero di rilascio della funzione. Ad esempio, JDK 20 .0.1 è installato in . `/Program Files/Java/jdk-<FEATURE><FEATURE>/Program Files/Java/jdk-20`

Verifichiamo se il JDK è installato correttamente dal prompt dei comandi digitando

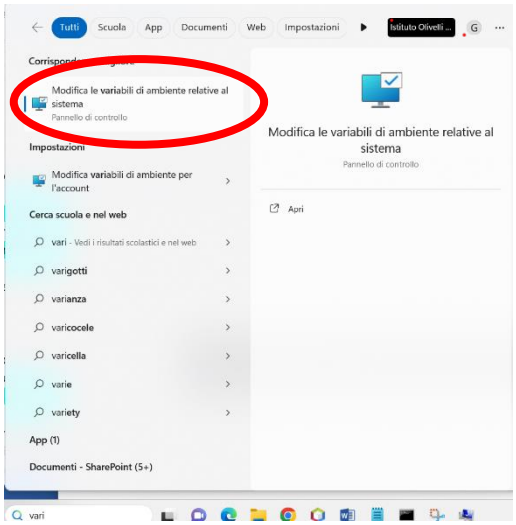
```
C:\Users\Gian>java -version
```

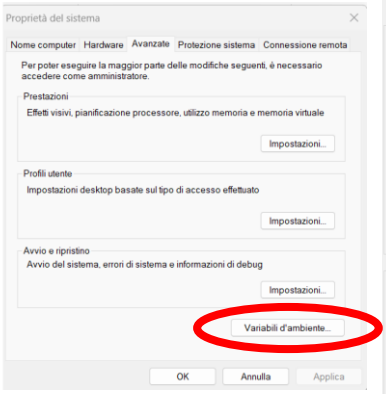
Apparirà il nome della versione del JDK installata

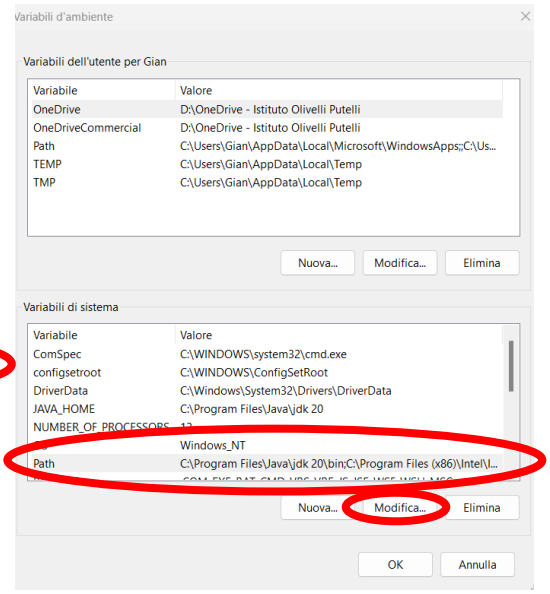
```
C:\Users\Gian>java -version
openjdk version "20" 2023-03-21
OpenJDK Runtime Environment (build 20+36-2344)
OpenJDK 64-Bit Server VM (build 20+36-2344, mixed mode, sharing)
```

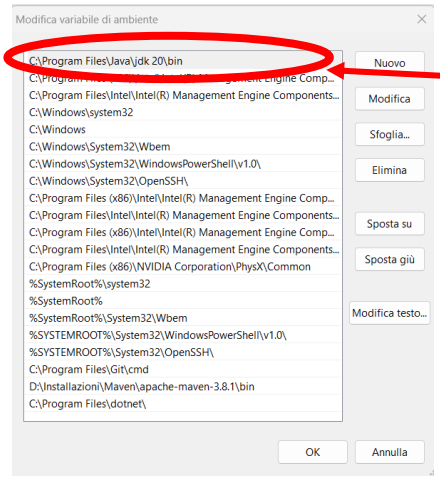
SUGGERMENTO IN CASO DI PROBLEMI: Può darsi che il comando "java -version" non venga riconosciuto. **Oppure che la versione di java in utilizzo non sia l'ultima che è stata installata**. Il motivo è che non è impostata correttamente la variabile d'ambiente path. La variabile d'ambiente "path" è una variabile che "dice" al sistema operativo in quale cartella "trovare" i file che consentono di avviare i vari programmi installati. Affinché il JDK funzioni, è necessario che nella variabile path sia presente il "percorso" della cartella contenente i file binari dell'ultima versione del JDK installata (nel nostro caso la cartella "C:\Program Files\Java\jdk 20\bin"). Durante l'installazione del JDK, l'installer dovrebbe opportunamente modificare tale variabile d'ambiente ma questo dipende dal tipo di installazione eseguita.

Per modificare la variabile d'ambiente path bisogna:









Modificare indicando la cartella bin dell'ultima versione del JDK installato

Pertanto, anche quando si vuole aggiornare il JDK con l'ultima versione, dopo averlo scaricato ed installato controlliamo che sia quella effettivamente in esecuzione e, nel caso non lo fosse, modifichiamo la variabile di ambiente path.

Per comprendere meglio il ruolo del compilatore e della JVM proviamo (solo per questa volta, poi installeremo un IDE chiamato NetBeans) a scrivere ed eseguire il primo programma java Hello.java da linea di comando, ma prima spieghiamo alcune cose:

1. L'elemento su cui si basa java è la classe, essa descrive come sono fatti gli oggetti che verranno istanziati, ed è costituita da due parti: gli attributi (parte informativa), e i metodi (operazioni che si possono fare sugli oggetti)
2. Un programma è, dunque, un insieme di classi da cui vengono istanziati oggetti. Ogni classe viene scritta come file di testo, ed è memorizzata in un **proprio file** a cui viene dato nome uguale a quello della classe ed estensione ***.java***
3. Il punto di inizio di un programma può essere il metodo main() di una qualsiasi delle classi presenti nel programma. Tutte le classi possono avere il metodo main(). La classe scelta come entry point del programma può avere qualsiasi nome ed è indicata come "***main class***". Il programmatore può scegliere quale sarà la "main class", l'esecuzione di un programma inizierà dal metodo main della main class. In NetBeans la main class viene impostata nel menu **Run/Set Project Configuration/Customize…/Run indicare la classe nel campo Main Class** (comunque, se non selezionata, viene chiesta all'avvio dell'esecuzione).
   E' opportuno che il metodo main della "***main class***" abbia sempre la seguente signature:

   public static void main (String[] args)

   Il metodo main della main class ha sempre le seguenti caratteristiche:
   - è pubblico: ossia può essere invocato da qualunque punto del codice
   - è statico: (vedremo meglio cosa significa)
   - non restituisce alcun valore di ritorno (void)
   - può avere dei parametri. Tali parametri sono un array di stringhe.

Se il sistema operativo non trova Javac nonostante sia stato installato, aggiungere la cartella del compilatore all'ambiente di lavoro del sistema operativo. Vedi qui come si fa: https://www.andreaminini.com/java/compilazione-esecuzione-programma-java

- Scriviamo con blocco note la classe Hello e salviamola come file "Hello.java" in una qualsiasi cartella

```
public class Hello                    // La classe deve avere lo stesso nome del file in cui viene salvata: classe Hello → file Hello.java
{
        public static void main(String[] args)
        {
                System.out.print("Ciao Pierone");
        }

}
```

- Rechiamoci con il prompt dei comandi nella cartella che contiene il file salvato (nel mio caso la cartella hello sul desktop) e compiliamo da riga di comando digitando:

  javac Hello.java

- La compilazione genera il file in **bytecode** nel file con nome uguale a quello della classe ed estensione. class, nel nostro caso : **Hello.class**
- Facciamo eseguire il bytecode alla JVM richiamando da riga di comando la JVM con il comando java:

java Hello

Se vogliamo passare come parametro la stringa da stampare sul monitor:

```
public class Hello
{
        public static void main(String[] args)
        {
                System.out.println(args[0]);
        }
}
```

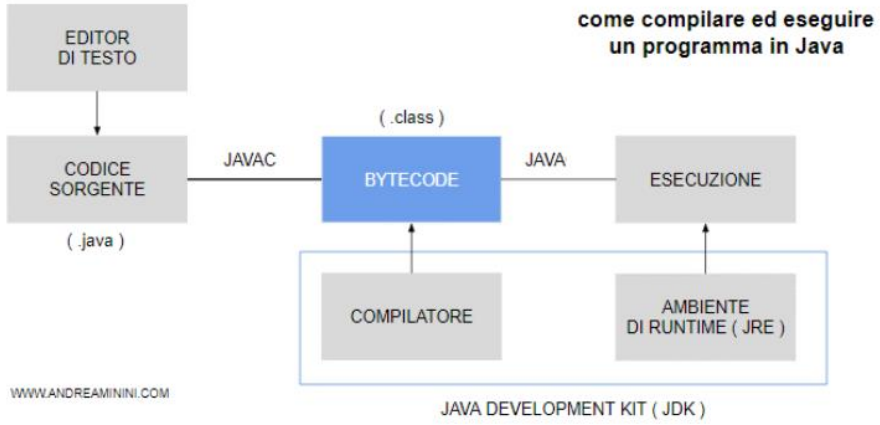
Dopo aver ricompilato si esegue nel seguente modo:

```
C:\Users\User\Desktop\Hello>java Hello "ciao ciao lao lao"
ciao ciao lao lao

C:\Users\User\Desktop\Hello>
```

La compilazione da riga di comando ci serve solo per vedere come avviene la compilazione, ora andiamo ad installare un IDE che ci aiuterà nella programmazione, pur continuando ad utilizzare il JDK per compilare.

Questo schema rappresenta in sintesi il processo di compilazione ed esecuzione di un programma Java.



3. L'IDE NetBeans

l'IDE che installiamo è Apache NetBeans versione 18, è un IDE open source che permette di scrivere programmi in Java, JavaScript, PHP, HTML 5,CSS ed altri linguaggi. Lo scarichiamo da qui (circa 500 Mb, l'installazione completa, **compresa la Enterprise Edition** richiederà circa 1GB di spazio sul disco):

https://netbeans.apache.org/

Scarichiamo ed installiamo la configurazione di default (nel nostro caso quella per windows x64). In questo video si può vedere passo passo l'installazione di NetBeans (la versione di NetBeans installata nel video è la 12.1, ma la procedura rimane la stessa anche per la versione NetBeans 18) e la realizzazione della prima classe Hello World:
https://www.youtube.com/watch?v=mDtbTtNgDJw&list=PLNrWrNHrd0qHhtrcR7KhBW_MPhNeX6u9&index=3

NetBeans va installato successivamente al JDK, infatti durante l'installazione NetBeans cerca (e solitamente trova, oppure ti chiede di indicare) la cartella in cui è installato il JDK da utilizzare.

Il software che utilizziamo è JAVA SE (Standard Edition). All'apertura dell'IDE viene proposto, fra le altre cose, un link ad un tutorial per applicazioni Java SE. Nel tutorial vengono mostrate le prime "scorciatoie" molto comode, da imparare:

- psvm +tab (public stastic void main): crea il metodo main
- sout+tab (system.output.print): crea il codice per scrivere sulla console standard di output (monitor)

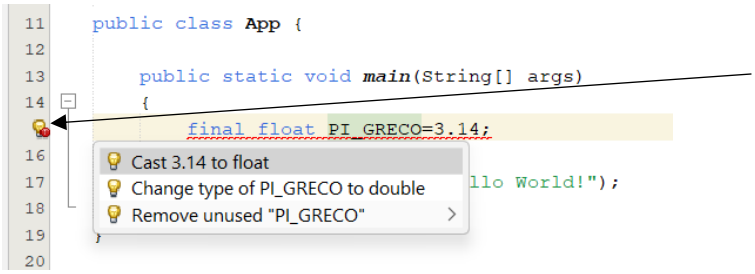
```
11  public class App {
12
13      public static void main(String[] args)
14      {
15
16          System.out.println(x: "Hello World!");
17      }
18  }
```

Cosa è questa x? è il nome del parametro della funzione printl(). E' un suggerimento dell'IDE. Puoi disattivarlo deselezionando i suggerimenti inline: view\Show Inline Hints

Dove vanno a finire i file creati durante il progetto? Partendo dal pathname della cartella di progetto, i file sorgenti (con estensione .java) si trovano nella cartella src\main\java all'interno delle sottocartelle che rappresentano i package (com\mycompany\.....), mentre i file in bytecode (estensione .class) si trovano nella cartella target\classes\ all'interno delle sottocartelle con il nome del package (com\mycompany\.....)

Svolgiamo alcuni esercizi (senza input) con osservazioni sulla sintassi e sull' IDE Net Beans. In questi primi esercizi non creeremo delle vere e proprie classi, creeremo una sola classe App (o Main) con un solo metodo main () e scriveremo del semplice codice per imparare la sintassi di Java come facevamo con la programmazione imperativa.

- 1_Cerchio Calcolare area di un cerchio dichiarando PI_GRECO come costante float di valore 3.14 e dichiarando le la variabile intera raggio e la variabile float area.
  In java, diversamente dal C, non esiste il preprocessore quindi non si può usare #define.
  Poiché il numero 3.14 viene inteso dal compilatore come un double, per assegnarlo ad una costante float servirà il casting esplicito. Il casting esplicito si può fare con la sintassi del C "(float)3.14" oppure ponendo una f al termine del numero "3.14f"
  Per dichiarare una costante si utilizza la parola chiave "final"
  - final float PI_GRECO =3.14f (chiede casting esplicito)



La sottolineatura indica un errore di sintassi. Cliccando sul pallino rosso a sinistra, oppure con **Alt+Enter,** appaiono i suggerimenti per risolvere l'errore. E' importante spere COME RISOLVERE l'eventuale errore e non accettare a priori la prima soluzione richiesta. **BISOGNA SAPERE COSA SI STA FACENDO!**

La combinazione **ctrl+space** svolge il completamento automatico del codice.

- 1_Cerchio: aggiungere calcolo del perimetro del cerchio scrivere il risultato concatenando 2 stringhe, per concatenare le stringhe si utilizza il il simbolo "+" all'interno della funzione println():

  System.out.println("l'area del cerchio e' " + area + "il perimetro del cerchio e' "+ perimetro).

- 2_Casting implicito, ipotizziamo che sia int raggio=5
  - println (raggio/2); cosa scrive?
  - println (raggio/2*1.0); cosa scrive? le operazioni vengono svolte da sx verso dx
  - println (1.0* raggio/2); cosa scrive?

- 2_Interi e caratteri:dualità int-char (e autoincremento):
  - attenzione, quando un short o un char vengono coinvolti in un' operazione essi vengono sempre riconvertiti in int, quindi se li vuoi in char è sempre necessario un casting esplicito.

```
char x='a';
char y=(char)(x+1);
System.out.println("il carattere successivo a "+x+" e'"+y);
```

Senza il casting non compila

Sorpresa (autoincremento postfisso, occhio!):

```
char x='a';
char y=x++;
System.out.println("il carattere successivo a "+x+" e'"+y);
```

La sintassi delle selezioni, dello switch (break, continue, default), e dei cicli è uguale a quella vista in C (svolgere autonomamente i seguenti esercizi):

- 3_Esercizio sconto piscina. Date le variabili (senza input utente, i valori alle variabili verranno assegnati nel codice) prezzo=10 € e tipoUtente, date le due costanti PERCENTUALE_SCONTO_DONNE

(=25) e PERCENTUALE_SCONTO_BAMBINI (50) , mostrare il costo dell'ingresso in piscina al variare del valore di tipoUtente (tipoUtente ='u', tipoUtente ='d', tipoUtente ='b')

- 4_Esercizio del mese

  Data una variabile intera "mese" a cui potranno essere assegnati (nel codice) i valori da 1 a 12, creare una struttura "switch case" che assegna ad una variabile "numeroGiorni" il numero di giorni di cui è costituito il mese in funzione della variabile "mese" (per semplicità consideriamo febbraio sempre con 28 giorni).

  Attenzione: per i case multipli la sintassi è

  case 4:
  case 6:
  case 9:
  case 11:
  numeroGiorni=30;

- 5_ Esercizio tabellina: stampare TRE VOLTE sul video la tabellina del 10 nel formato mostrato di seguito (solo stampare, non serve una matrice). La prima volta usando un doppio ciclo for e moltiplicando fra loro gli indici i e j dei cicli, la seconda usando un doppio ciclo do—while, la terza usando un doppio ciclo while.
  Ricordo che **System.out.println("…")** manda a capo la riga successiva, mentre
  System.out.print("…") non manda a capo la riga successiva.

```
Ciclo For
   1   2   3   4   5   6   7   8   9  10
   2   4   6   8  10  12  14  16  18  20
   3   6   9  12  15  18  21  24  27  30
   4   8  12  16  20  24  28  32  36  40
   5  10  15  20  25  30  35  40  45  50
   6  12  18  24  30  36  42  48  54  60
   7  14  21  28  35  42  49  56  63  70
   8  16  24  32  40  48  56  64  72  80
   9  18  27  36  45  54  63  72  81  90
  10  20  30  40  50  60  70  80  90 100
Ciclo While
   1   2   3   4   5   6   7   8   9  10
   2   4   6   8  10  12  14  16  18  20
   3   6   9  12  15  18  21  24  27  30
   4   8  12  16  20  24  28  32  36  40
   5  10  15  20  25  30  35  40  45  50
   6  12  18  24  30  36  42  48  54  60
   7  14  21  28  35  42  49  56  63  70
   8  16  24  32  40  48  56  64  72  80
   9  18  27  36  45  54  63  72  81  90
  10  20  30  40  50  60  70  80  90 100
Ciclo Do While
   1   2   3   4   5   6   7   8   9  10
   2   4   6   8  10  12  14  16  18  20
   3   6   9  12  15  18  21  24  27  30
   4   8  12  16  20  24  28  32  36  40
   5  10  15  20  25  30  35  40  45  50
   6  12  18  24  30  36  42  48  54  60
   7  14  21  28  35  42  49  56  63  70
   8  16  24  32  40  48  56  64  72  80
   9  18  27  36  45  54  63  72  81  90
  10  20  30  40  50  60  70  80  90 100
```

CONSIGLIO MOLTO UTILE PER OGNI STUDENTE: CREARE UN FILE EXCEL IN CUI PRENDERE GLI APPUNTI SULLE COSE NUOVE VISTE IN OGNI ESERCIZIO

CONSIGLIO 2: CREARE UN DATABASE CONDIVISO DEGLI ERRORI DI COMPILAZIONE. LO CREIAMO INSIEME A LEZIONE POI OGNUNO LO PUO' COMPILARE NEL TEMPO

Dopo questi primi esercizi, ora realizziamo una vera e propria classe, la classe Televisore di cui abbiamo parlato nella parte sulla introduzione alla OOP.

Risorsa video per questa lezione: **https://www.youtube.com/watch?v=CuATaQsMTgw&list=PL-NrWrNHrd0qHhtrcR7KhBW_MPhNeX6u9&index=5**

Nella programmazione ad oggetti uno dei concetti principali è il concetto di **classe**

Una classe rappresenta un modello formale per la descrizione di un certo tipo di oggetti definendone gli **attributi** (la componente informativa), i **metodi** (le operazioni che possono essere svolte su quel tipo di oggetti) e **l'interfaccia** (l'insieme di metodi attraverso i quali è possibile interagire con quel tipo di oggetti).

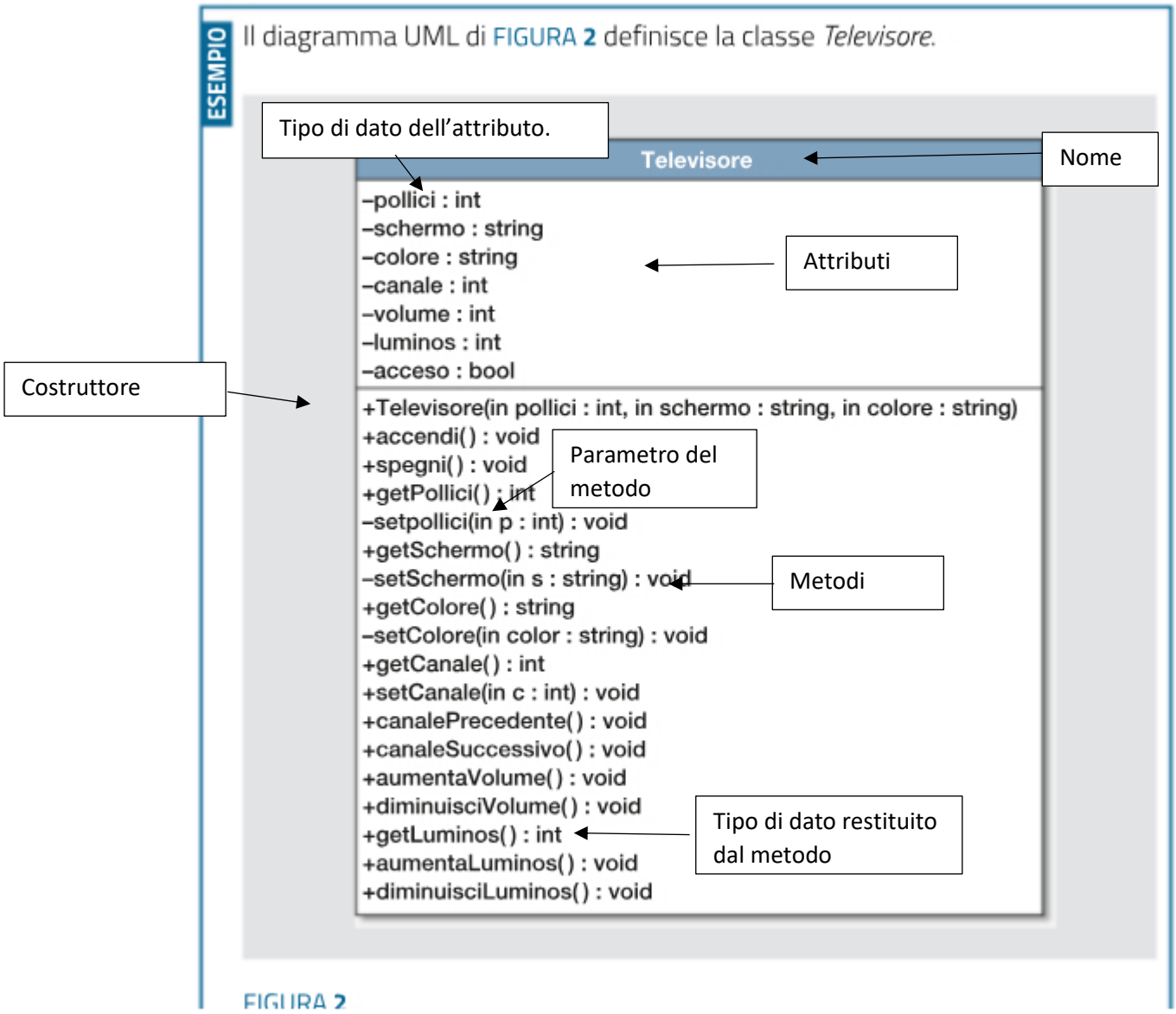
In un programma OOP, da una classe possono essere **istanziati** (creati) degli oggetti. Si può dire che la classe è una descrizione (un modello) di un oggetto, mentre l'oggetto è un esemplare di quella classe.

Quando, durante l'esecuzione di un programma, viene istanziato un oggetto succedono due cose:

1. Viene allocato dello spazio nella memoria centrale in cui memorizzare tale oggetto
2. Vengono inizializzati gli attributi dell'oggetto.

Nel programma, la creazione di un oggetto avviene invocando un metodo particolare della classe stessa chiamato **costruttore,** tale metodo è un metodo sempre presente in una classe, ha lo stesso nome della classe (Il costruttore della classe Televisore si chiama Televisore(….)), non restituisce alcun tipo di dato, ha dei parametri che permettono l'inizializzazione di un oggetto di quella classe.

Per rappresentare graficamente le classi e le relazioni fra di esse si utilizza uno specifico diagramma, chiamato **diagramma delle classi**, che fa parte dell'insieme dei diagrammi **UML** (Unified Modeling Language). Vediamo il diagramma delle classi per la classe Televisore.



Si osservi che i può usare il "tipo di dato" **String** (in realtà è essa stessa una classe, poi spiegheremo meglio) che permette di creare variabili e attributi a cui vengono assegnati delle stringhe alfanumeriche.

I + e – davanti ai metodi e agli attributi indicano se tali elementi sono pubblici o privati:

+ : l'elemento è pubblico. Quando il programmatore istanzia un oggetto di classe Televisore, potrà accedere a quell'elemento (ricordo che la parola "elemento" indica un attributo o un metodo).

- : l'elemento è privato. Se nel codice di un'altra classe verrà istanziato un oggetto di classe Televisore, NON si potrà accedere a quell'elemento (attributo o metodo).

I metodi pubblici costituiscono **l'interfaccia** della classe, infatti sono gli unici metodi che consentono ad un programmatore di interagire con l'oggetto istanza di quella classe.

GENERALMENTE, tranne in casi particolari, nella costruzione delle classi si seguono le seguenti regole:

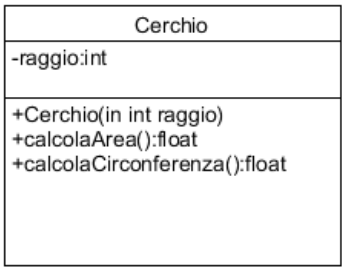
1. Tutti gli attributi sono privati (per proteggerli)
2. Il costruttore può avere dei parametri che consentono al programmatore che vuole istanziare un oggetto, di inizializzare nel modo desiderato l'oggetto.

3. Per consentire di modificare lo stato interno di un oggetto, ossia di modificare il valore di **alcuni** attributi, la classe presenta dei metodi particolari il cui nome è "get+nome attributo" oppure "set+nome attributo", tali metodi sono chiamati "**getter**" e "**setter**".
4. I **getter** consentono di "leggere" il valore di un attributo.
   I **setter** consentono di modificare (scrivere) il valore di un attributo.

Esercizio Figure_geometriche:

Creiamo insieme la classe cerchio e istanziamo un oggetto di classe Cerchio con raggio 10:

Diagramma della classe



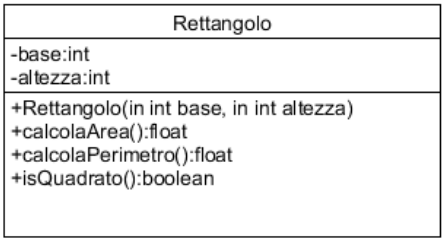
Codice della classe cerchio

```
11  public class Cerchio
12  {
13      private int raggio;
14
15      public Cerchio(int raggio)
16      {
17          this.raggio=raggio;
18      }
19
20      public float calcolaArea()
21      {
22          return (float)3.14*raggio*raggio;
23      }
24
25      public float calcolaCirconferenza()
26      {
27          return (float)3.14*2*raggio;
28      }
29  }
30
```

Codice della main class in cui viene istanziato un oggetto di classe cerchio chiamato C1 con raggio 10 e ne viene calcolata area e circonferenza invocandone gli appositi metodi

```
11  public class App {
12
13      public static void main(String[] args)
14      {
15          Cerchio c1=new Cerchio(raggio:10);
16          System.out.println("C1\narea:"+c1.calcolaArea()+"\ncirconferenza:"+c1.calcolaCirconferenza());
17      }
18  }
19
```

Lo studente aggiunga da solo la classe rettangolo all'esercizio precedente. Nella classe App (la Main class) si istanzino un oggetto cerchio di raggio 20 e due oggetti rettangoli con base ed altezza a piacere. Si visualizzino area e perimetro di tutti e tre gli oggetti. Inoltre si comunichi sulla console, per ogni rettangolo, se esso è un quadrato o meno utilizzando il metodo isQuadrato.



5. GLI ELEMENTI LOGICI CHE COSTITUISCONO LA JVM

Risorsa video per questa lezione: https://www.youtube.com/watch?v=4m4BmDp9pE8&list=PL-NrWrNHrd0qHhtrcR7KhBW_MPhNeX6u9&index=7

Come detto possiamo vedere la JVM come un "computer virtuale" che "esegue" il bytecode. (L'esecuzione del bytecode, nella realtà, consiste nella sua traduzione in binario e nell' esecuzione da parte del processore reale).

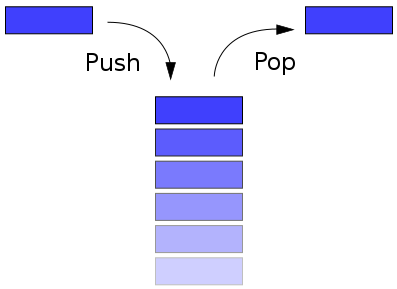
La JVM è un "computer virtuale" che esegue il byte code, ed è dunque formata dai seguenti elementi virtuali, che le consentono di eseguire il bytecode. Questi elementi virtuali, in realtà non sono altro che zone della memoria centrale in cui sono memorizzati dei valori binari. Questi elementi si chiamano: registri, area dei metodi, stack, heap.

- **Registri**: come in un processore reale, i registri determinano lo stato in cui si trova la macchina durante l'esecuzione del bytecode (program counter, instruction register, MAR, DAR..)
- **Area dei metodi**: contiene il codice in bytecode dei metodi delle classi del programma in esecuzione, le istruzioni da eseguire
- **Stack**: è un'area di memoria centrale utilizzata come area di lavoro durante l'esecuzione dei metodi. In quest'area vengono memorizzate tutte le variabili utilizzate durante l'esecuzione di un metodo: variabili locali (ad esempio la variabile "raggio" del cerchio), parametri dei metodi, valori di ritorno dei metodi. Si ricorda che i metodi svolgono lo stesso ruolo che nella programmazione imperativa svolgono le funzioni.

  Lo stack va compreso bene. Con il termine stack, in informatica, si definisce, una struttura dati di tipo LIFO (Last In First OUT), detta anche "pila" (come la pila dei piatti del ristorante). In questa struttura i dati sono quindi memorizzati uno sopra l'altro. Su di uno stack si possono svolgere due operazioni: PUSH e POP.

  Operazione PUSH: Inserimento di un dato nello stack, il dato inserito si sovrappone a quelli presenti.

  Operazione di POP: estrazione di un dato dallo stack, il dato estratto è l'ultimo che è stato inserito.



  Lo stack è una struttura dati utilizzata in diversi ambiti dell'informatica. Ora a noi interessa in particolare il suo utilizzo legato alla chiamata dei metodi.

  Ogni volta che viene invocato un metodo o funzione (e il main della main class è anch'esso un metodo), succede che in **una zona prestabilita della RAM** (lo stack, appunto) vengono riservate delle celle per memorizzare i parametri e le variabili locali del metodo invocato.

  Tale zona di memoria è detta appunto stack. In realtà vengono memorizzate nello stack anche altre informazioni oltre a variabili locali e parametri, le informazioni che sono necessarie per riprendere

l'esecuzione del metodo chiamante una volta che è terminata l'esecuzione del metodo chiamato (principalmente: i valori dei registri del processore e l'indirizzo di ritorno).

L'entry point di un programma è il metodo main della main class. Il codice del main, e degli altri metodi, si trova "memorizzato da qualche parte" nella memoria centrale. Quando viene invocato un metodo "A" (o una funzione), il processore esegue un "salto" fra gli indirizzi della RAM, ossia va a leggere all' indirizzo in cui si trova la prima istruzione del metodo "A". Prima di fare questo "salto" il processore memorizza in una parte della memoria, lo stack appunto, (non ci interessa dove) i dati che gli consentiranno, in seguito, di riprendere l'esecuzione "dal punto giusto" (i dati sono memorizzati sono: valori delle variabili, dei parametri, indirizzo dell'istruzione di ritorno ecc.). Terminata l'esecuzione del metodo "A", il processore ritornerà a leggere all'indirizzo in cui si trovano le istruzioni del metodo main, per eseguire il codice dal punto in cui era stato interrotto.

Ogni metodo può, a sua volta, richiamare altri metodi, sempre memorizzando parametri, variabili locali e informazioni per il ritorno. L' indirizzo dell'ultima istruzione inserita nello stak, quindi la prima da estrarre, è sempre memorizzata in un puntatore chiamato stack pointer. Tutto quanto è stato descritto, avviene in maniera trasparente al programmatore (lui non se ne accorge), ma è importante saperlo, per capire cosa è, ad esempio, uno **stack overflow**, che può avvenire quando una funzione ne richiama molte altre (migliaia) o richiama se stessa migliaia di volte (cosa che avviene ad esempio in una particolare struttura di programmazione chiamata ricorsione).

Vedi file power point sullo stack.

Osservazione importante:

1. Lo stack viene **allocato dinamicamente** (ossia durante l'esecuzione del programma vengono allocate/deallocate zone di memoria centrale destinate agli stack frame)
2. Ogni volta che il SO deve allocare uno stack frame per una funzione "ha bisogno" di sapere quante e che tipo di variabili sono dichiarate nella funzione, questo è necessario affinchè possa allocare la quantità di memoria corretta (non troppa, che andrebbe sprecata, non troppo poca, che non sarebbe sufficiente a contenere i dati necessari).
3. Per questo motivo è sempre obbligatorio che il programmatore specifichi il numero e il tipo di variabili prima di utilizzarle in qualsiasi metodo (compreso il metodo main della main class).
4. Per questo motivo quando usiamo gli array siamo costretti ad indicare in fase di scrittura di un programma, la dimensione degli array. Questo è ciò che abbiamo sempre fatto, ad esempio nel programma "Gestione Studenti" abbiamo dichiarato un array di 100 studenti (vuoto) che poi andavamo a riempire di valori man mano che venivano inseriti nuovi studenti.

Problema:

1. In un software reale non è sempre possibile conoscere a priori la quantità di dati da memorizzare. Ad esempio: se gli studenti sono più di 100? ….potrei creare un array da 1000 studenti. E se gli studenti sono 10000? Non si può stabilire a priori un numero massimo di studenti, quindi bisognerebbe dichiarare un array di grandi dimensioni (1000000 ? boh?). Ma se poi il software viene utilizzato in una scuola che ha solo 100 studenti? Ci sarebbe un grande spreco di memoria (il compilatore riserva lo spazio in memoria per TUTTI gli elementi dell'array, indipendentemente da quanti poi saranno gli elementi effettivamente utilizzati).

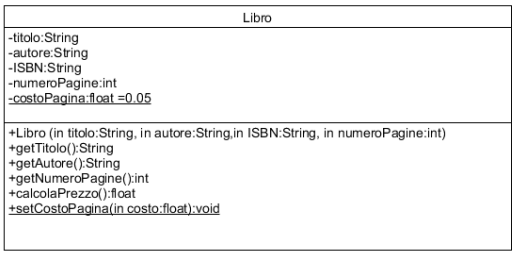
Soluzione:

1. Per risolvere questo problema sarebbe opportuno poter avere delle strutture dati **allocate dinamicamente**, in cui il numero di elementi può aumentare o diminuire durante

l'esecuzione del programma, a run time. Questo è ciò che è possibile fare nella zona di memoria centrale chiamata Heap.

- **Heap**: zona di memoria **allocata dinamicamente** in cui vengono memorizzati gli **oggetti** istanziati dalle classi con la keyword **new** In C/C++ l'allocazione dinamica si ottiene utilizzando i puntatori (variabili che contengono indirizzi di memoria), in java la gestione dinamica della memoria è semplificata rispetto al C/C++ (i puntatori sono mascherati, si usano ma non si vedono direttamente). La memoria necessaria per creare gli oggetti istanziati è dunque allocata dinamicamente in una apposita zona della RAM diversa dallo stack chiamata heap. Grazie all'allocazione dinamica è possibile creare strutture dati (ad esempio le liste) in cui il numero di elementi può crescere indefinitamente a run time. Senza bisogno di definire un numero massimo di elementi, il programmatore potrà definire un metodo "aggiungiElemento()" che, ogni volta aggiungerà un elemento alla lista. Tale metodo potrà essere invocato, su richiesta dell'utente, e ogni volta andrà ad aggiungere un elemento alla lista, senza limiti, se non quelli imposti dalla memoria fisica.

  Osservazione importante: in una classe ci possono essere degli attributi che hanno lo stesso valore in tutti gli oggetti che verranno istanziati da quella classe, tali attributi sono detti **statici** (questo termine viene usato ogni volta che ci si riferisce a qualcosa relativo all'intera classe e non ai singoli oggetti istanziati da quella classe) per distinguerli da quelli che invece assumeranno valori diversi per ogni singolo oggetto, detti **non statici**. Quando vengono istanziati degli oggetti di una classe con attributi statici, nello heap il valore degli attributi statici verrà memorizzato una sola volta, visto che assume lo steso valore per ogni oggetto. Ad esempio, data una classe "Libro" si potrebbero definire l'attributo statico costoPagina che indica il costo necessario per produrre ogni pagina dei un libro e serve per determinare il prezzo di vendita del libro.

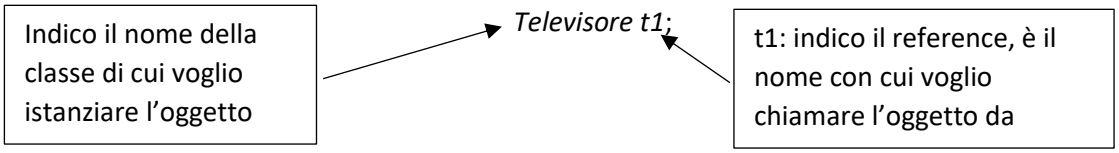


  Gli attributi statici (e anche i metodi che agiscono su attributi statici) sono rappresentati nel diagramma delle classi con una sottolineatura.
  Nell'esempio, ogni libro istanziato avrà un valore di costo pagina =0,05, e se un giorno si decidesse di modificare tale attributo invocando il metodo "setCostoPagina(0,1)" su un'istanza qualsiasi, il valore dell'attributo "costoPagina" cambierebbe in tutti i libri istanziati. Con il termine **statico** si intende qualcosa che si riferisce alla classe e quindi "**non può modificarsi nel singolo oggetto**" ma solo in tutta la classe e di conseguenza in tutti gli oggetti istanza di quella classe.

COME ISTANZIARE OGGETTI IN JAVA (IMPORTANTISSIMO)

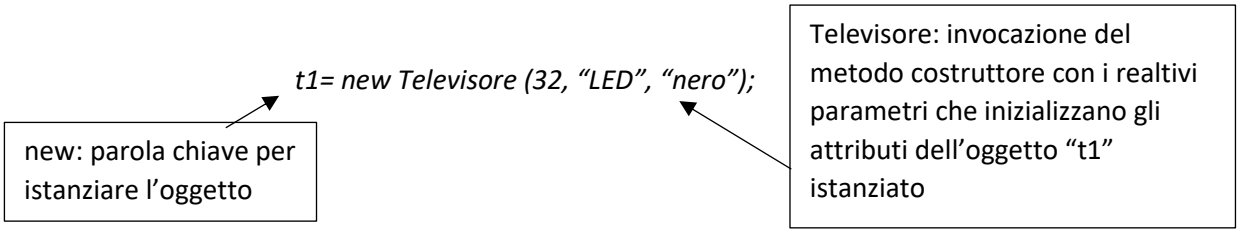
Risorsa video per questa lezione: https://www.youtube.com/watch?v=DFdunMdAfJQ&list=PL-NrWrNHrd0qHhtrcR7KhBW_MPhNeX6u9&index=9

Come detto, dalle classi vengono istanziati gli oggetti. Ad ogni oggetto è associato un **identificatore** (come per le variabili), ad esempio dalla classe Televisore possiamo istanziare un oggetto e chiamarlo t1:



Quando viene eseguita questa istruzione, ancora non è stato istanziato l'oggetto, ma solamente l'identificatore "t1", la JVM riserva spazio nello stack per una "variabile" che può contenere l'indirizzo in memoria di un oggetto di classe Televisore, ma ancora l'oggetto non è stato creato.

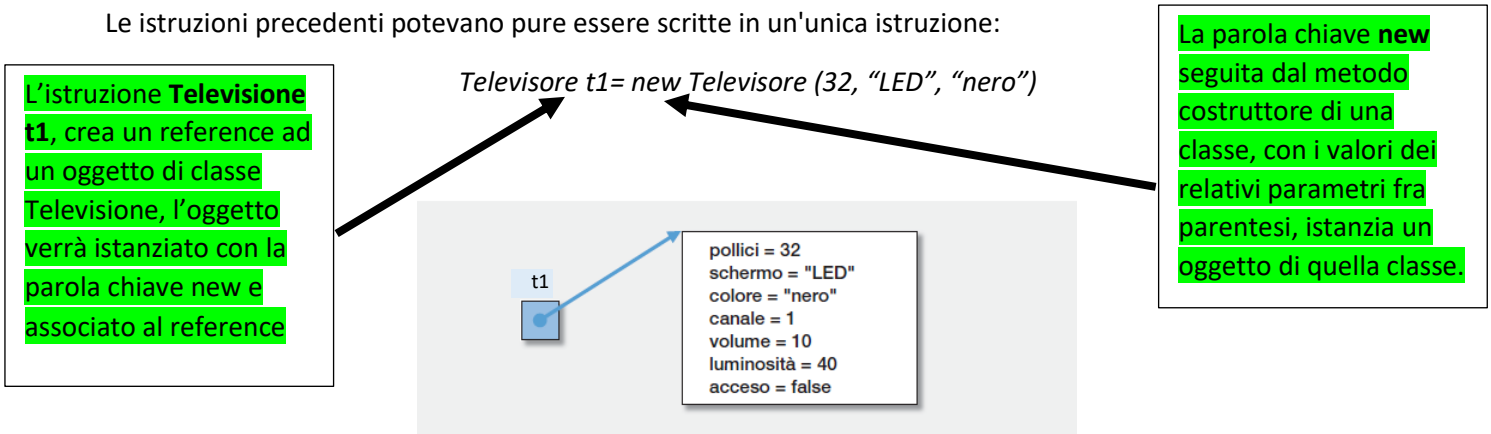
Per istanziare l'oggetto è necessario che venga **eseguita** l'istruzione **new**:



Solo QUANDO VIENE ESEGUITA QUESTA ISTRUZIONE (a **runtime**) verrà istanziato un oggetto di classe televisore (**nello heap**) e t1, in memoria, conterrà l'indirizzo a cui si trova questo oggetto in memoria (nello heap).

In pratica t1 è un **puntatore mascherato** (o meglio, puntatore implicito), detto anche **riferimento** o **reference**, ossia una variabile (si trova nello stack) che contiene l'indirizzo della memoria (heap) in cui si trova l'oggetto t1 creato. Prima dell'esecuzione di questa istruzione, la variabile t1 conteneva il valore **NULL** (valore assegnato ad un "puntatore" che non punta a nessuna locazione di memoria).

Le istruzioni precedenti potevano pure essere scritte in un'unica istruzione:



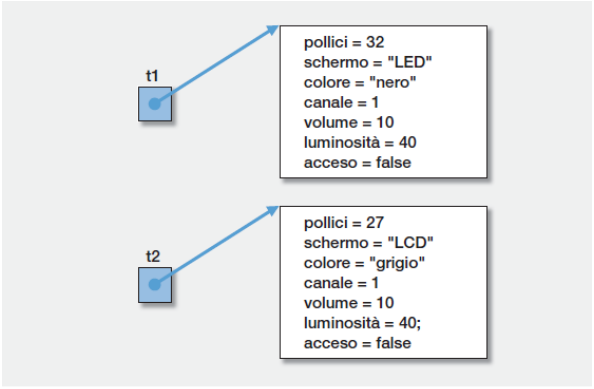
ora è possibile, nel codice, utilizzare il riferimento t1 per *agire* sull' oggetto televisore da esso puntato. Per agire sul televisore t1 si utilizzano i metodi che la classe espone nell'interfaccia (alzaVolume(), canaleSuccessivo(), ecc..)

Ciò che succede nella memoria centrale è questo:

Se istanziamo due oggetti televisore abbiamo bisogno di due reference, t1 e t2, ecco cosa succede:

Televisore t1= new Televisore (32, "LED", "nero")

Televisore t2= new Televisore (24, "LED", "grigio")



Attenzione: cosa succede con la seguente istruzione?

t2=t1;

t2 è un puntatore, anch'esso ora punta a allo stesso televisore a cui punta t1, (quello nero)!

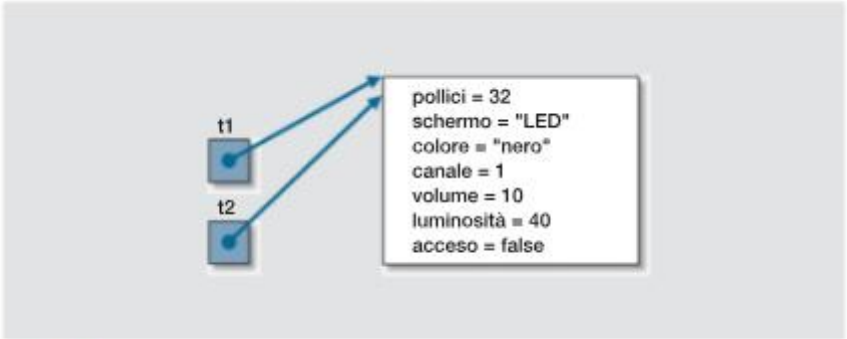


FIGURA 4

E il televisore precedentemente puntato da t2 che fine ha fatto (quello grigio)? è ancora nello heap? La risposta ovvia sarebbe SI, e invece NO! Perché? dobbiamo spiegare il ***garbage collector*** (raccoglitore di spazzatura).

In java esiste questo strumento chiamato **garbage collector** che viene periodicamente (e automaticamente) attivato e che serve ad eliminare tutti gli oggetti che sono rimasti nello Heap ma che sono privi di riferimento, ossia non hanno alcun reference che li punta. Tali oggetti infatti occupano inutilmente spazio nella RAM poiché sono inutilizzati. La presenza di "spazzatura" avviene, ad esempio, quando all'interno dei metodi vengono creati degli oggetti (che hanno dei reference locali nel metodo) che poi, alla chiusura del metodo non hanno piu' reference associati.

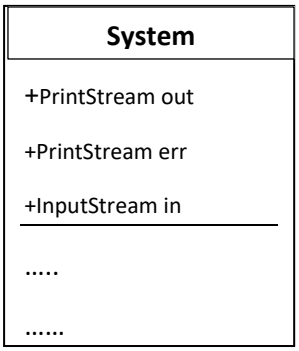
Il garbage Collector è uno strumento molto utile, in C/C++ non esiste, ed infatti ogni volta che un oggetto non viene più utilizzato deve essere il programmatore, con un apposito comando che ne dealloca lo spazio occupato in memoria (invece in C# è presente il garbage collector)

4. OUTPUT, INPUT E NUMERI CASUALI IN JAVA

Risorsa video per questa lezione: https://www.youtube.com/watch?v=4On3EnpQkxU&list=PL-NrWrNHrd0qHhtrcR7KhBW_MPhNeX6u9&index=8
Per comunicare sulla console di output si utilizza la seguente istruzione:

System.out.println("Hello");

Spieghiamola meglio.
System è una **classe astratta**. Una classe astratta è una classe che non può essere istanziata, cioè dalla quale non si possono istanziare oggetti, ma ha metodi statici che si possono invocare direttamente sulla classe. La classe System, che java mette a disposizione del programmatore per operazioni di I/O, ha diversi attributi e metodi statici:



Out è un attributo (statico) di System. Di che tipo è?
Gli attributi possono essere di tipo nativo (**int, double, float, ecc..)** ma anche oggetti di una classe. Nel nostro caso dunque fra gli attributi della classe System ce n'è uno chiamato **out** che è un'oggetto, istanza della classe **PrintStream**.

A sua volta quindi out, in quanto istanza di una classe, avrà attributi e metodi della classe PrintStream.
In particolare il metodo **println** (e **print**, che è uguale ma non va a capo) è un **metodo** della classe PrintStream che consente di scrivere sulla console di output.

Scorciatoia NetBeans: *outs+tab*

Come parametri, il metodo println() accetta delle stringhe o dei valori numerici. Le stringhe possono essere concatenate con il simbolo +. Quando vengono concatenati oggetti che sono tipi di dato predefiniti (float int ecc.., poi li vedremo tutti) o oggetti istanza di classi che implementano il metodo *toString* (che si usa solitamente per rappresentare con una stringa gli attributi di un oggetto), la concatenazione "**+**" fa sì che il metodo *toString* venga **automaticamente** invocato.

Le stringhe **sono oggetti della classe String** e sono sempre scritte fra virgolette. All'interno delle stringhe sono accettate le seguenti sequenze di escape:

Escape Sequence	Character
\n	newline
\t	tab
\b	backspace
\f	form feed
\r	return
\"	" (double quote)
\'	' (single quote)
\\	\ (back slash)
\uDDDD	character from the Unicode character set (DDDD is four hex digits)

Carattere utilizzato per indicare la fine di una pagina di testo. Veniva utilizzato nelle stampanti per forzare la fuoriuscita del foglio.

Sequenza esadecimale che rappresenta un carattere UNICODE. La console potrebbe non essere in grado di visualizzare tale carattere. Prova con System.out.println("ciao Pierone \u00a9");

00A9 → ©

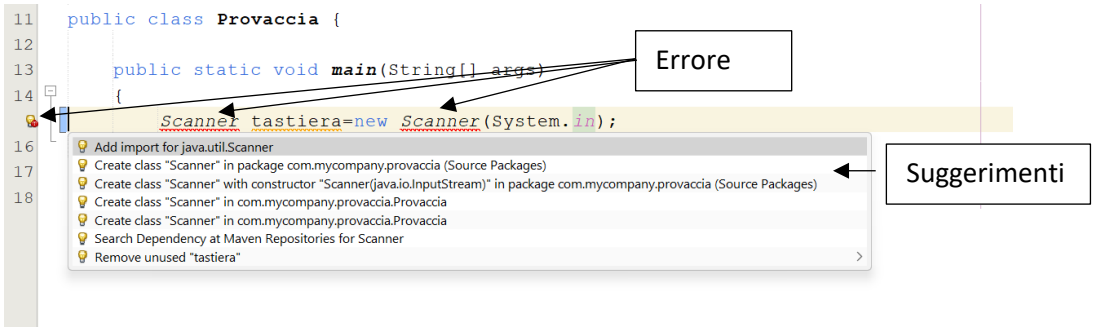
Input da tastiera

Risorsa video per questa lezione: https://www.youtube.com/watch?v=4On3EnpQkxU&list=PL-NrWrNHrd0qHhtrcR7KhBW_MPhNeX6u9&index=8

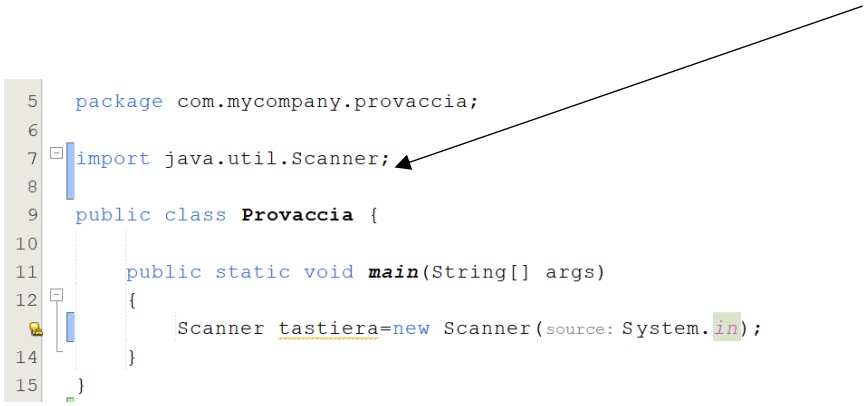
Ci sono due modi per leggere da tastiera. Ora ne mostriamo uno (classe **Scanner**).
Per leggere creiamo un oggetto "tastiera" istanza della classe ***Scanner*** all'interno del metodo main() di una classe di prova qualsiasi.
Per utilizzare la classe Scanner è necessario importare il package ***java.util.Scanner (o meglio java.util.*)***

SUGGERIMENTO: per importare un package necessario in automatico con netBeans si può ricorrere ai suggerimenti dell'IDE, infatti quando c'è un errore di sintassi (ad esempio il package non è stato importato) NetBeans evidenzia il codice con una sottolineatura e con una "lampadina" in corrispondenza del numero di linea.
A questo, punto cliccando sulla lampadina, NetBeans "suggerisce" alcune possibili soluzioni per l'errore. L'errore di sintassi viene risolto cliccando sulla soluzione opportuna fra quelle suggerite. E' importante non cliccare a caso questi suggerimenti ma capire bene cosa significano e cliccare sul suggerimento adatto al nostro codice. Nel caso di mancato import del package la soluzione corretta è "Import package…". Cliccando tale soluzione il package mancante verrà importato nel codice sorgente.

Esempio



Ciccando sul primo suggerimento, la classe Scanner viene importata automaticamente dal package “java.util”



```
5   package com.mycompany.provaccia;
6
7   import java.util.Scanner;
8
9   public class Provaccia {
10
11      public static void main(String[] args)
12      {
13          Scanner tastiera=new Scanner(source: System.in);
14      }
15  }
```

Come parametro all’oggetto Scanner dobbiamo indicare la provenienza dei dati da leggere (la tastiera) che si indica con l’oggetto: ***System.in.***
Per leggere il tipo di dato (intero, double, stringa ecc…) si utilizzano gli appositi metodi (nextInt, nextdouble, nextString…) dell’oggetto creato “tastiera” di tipo Scanner. Quindi:

```
Scanner tastiera= new Scanner(System.in);
prezzo=tastiera.nextInt();
```

Per ora ipotizzeremo negli esercizi che i dati inseriti siano sempre corretti, ossia evitiamo il controllo dell’input.
I metodi di Scanner che useremo sono:

nextInt(), nextDouble, nextFloat
nextLine()→ Legge una stringa fino al return

Esercizio 1: numero massimo. Si svolga l'esercizio nel metodo main di una qualsiasi classe java di prova.
Risorsa video per questa lezione: https://www.youtube.com/watch?v=lWP9GmWAmmM&list=PL-NrWrNHrd0qHhtrcR7KhBW_MPhNeX6u9&index=10

Scrivere un programma in java che

- acquisisca due numeri interi da tastiera.
- confronti i due numeri ed indichi quale dei due è il maggiore
- chieda ripetutamente se l'utente vuole inserire un altro numero e ogni volta comunichi quale è il maggiore fra tutti i numeri inseriti. (Si usi una variabile intera come "continua" 1→continua, 0→ stop).

Generare numeri casuali in Java

Risorsa video per questa lezione: https://www.youtube.com/watch?v=f1Mkpx_Y_3g&list=PL-NrWrNHrd0qHhtrcR7KhBW_MPhNeX6u9&index=11

Per generare un numero pseudo casuale si utilizza la classe **Random**. Tale classe deve essere importata dal package **java.util.Random.**
Un numero pseudo casuale è un numero che viene determinato mediante un algoritmo a partire da un numero di input di tipo long chiamato seme (**seed**).
Il seed può essere passato come parametro al costruttore della classe Random.
A partire dallo stesso seme la classe random genererà sempre la stessa sequenza di numeri.
Per generare numeri casuali interi si invoca il metodo **nextInt()** sull'oggetto istanziato dalla classe **Random**

Esempio di utilizzo 1: Genera una sequenza di 10 numeri pseudo casuali interi (4 byte). Ogni volta che il codice viene eseguito la sequenza si ripete.

```
public static void main(String[] args)
  {
    Random r=new Random(0);
    for (int i=0;i<10;i++)
    {
      System.out.println(r.nextInt());
    }
  }
```

r= oggetto istanza della classe Random

seed=0

r.nextInt() genera il numero pseudo-casuale

Esempio di utilizzo 2: Generare una sequenza di numeri casuali compresi fra 0 e 10. Ogni volta che il codice viene eseguito la sequenza si ripete.
Per limitare l'intervallo di numeri pseudo casuali generati fra 0 e un certo limite (escluso) si passa tale "limite" come parametro al metodo nextInt(). Un valore limite è generalmente chiamato **bound**.

```
public static void main(String[] args)
  {
    Random r=new Random(0);
    for (int i=0;i<10;i++)
    {
      System.out.println(r.nextInt(11));
    }
  }
```

bound=11

verranno generati numeri pseudo casuali fra 0 (compreso) e 10 (compreso)

Esercizio da fare: come faccio a generare una sequenza di numeri pseudo casuali compresi fra 0 e 90? Farlo.
Fare: si verifichi che cambiando il seed, la sequenza dei numeri cambia

Esempio di utilizzo 3: Genera una sequenza, ogni volta diversa, di numeri pseudo casuali interi compresi fra -18 e 36. Ogni volta che il codice viene eseguito la sequenza cambia.
Per impostare un intervallo con un valore minimo (-18) e un valore massimo (36) basta aggiungere il valore minimo (-18) e il valore massimo aumentato di 1 (37) come parametri del metodo nextInt():

```
public static void main(String[] args)
  {
    Random r=new Random();
    for (int i=0;i<10;i++)
    {
      System.out.println(r.nextInt(-18,37));
    }
  }
```

Il seed viene impostato automaticamente ed è diverso per ogni esecuzione

Il primo parametro indica il valore minimo, il secondo parametro indica il bound.

Esercizio:

Scrivere un programma java che genera un numero casuale compreso fra 1 e 10.
Il programma chiede ripetutamente all'utente di indovinare il numero estratto fino a che l'utente decide di terminare il gioco oppure indovina il numero. (Usare una variabile intera come "continua" 1→continua, 0→ stop).
Poi modificare il gioco in modo che vengano conteggiati i tentativi fatti prima di indovinare il numero.

```
package com.mycompany._indovina_il_numero;
import java.util.*;
public class Main
{
   public static void main(String[] args)
   {
      Random r=new Random();
      Scanner tastiera= new Scanner (System.in);

      int numeroDaIndovinare;
      int tentativo;
      int continua=1;

      numeroDaIndovinare=1+r.nextInt(10);

      do
      {
         System.out.println("Quale numero è stato estratto?");
         tentativo=tastiera.nextInt();
         if (tentativo==numeroDaIndovinare)
            System.out.println("Hai vinto!");
         else
         {
            System.out.println("Sbagliato, cuoi tentare di nuovo? 1--SI altro-->NO");
            continua=tastiera.nextInt();
         }
      }while (continua ==1 && numeroDaIndovinare!=tentativo);

   }
}
```

Esercizio da fare: esercizio dadi (vedi apposito file)

Risorsa video parte 1: https://www.youtube.com/watch?v=Kr7nx9xlOSc&list=PL-NrWrNHrd0qHhtrcR7KhBW_MPhNeX6u9&index=12
Risorsa video parte 2: https://www.youtube.com/watch?v=CBjSn91uU4U&list=PL-NrWrNHrd0qHhtrcR7KhBW_MPhNeX6u9&index=13