Corso di fondamenti di informatica – seconda parte

Ingegneria industriale - polo di Frosinone

Indice

Le caratteristiche generali di Java

Java è un linguaggio ad alto livello e orientato agli oggetti. Java è nato all'interno di un progetto di ricerca molto ampio che aveva lo scopo di creare software avanzato per vari sistemi. Inoltre è stato pensato in riferimento all'importanza che hanno assunto le reti di computer negli ultimi anni e ha avuto una forte integrazione con Internet, in particolare con la possibilità di scrivere piccole applicazioni per la rete (applet), eseguibili all'interno delle pagina web visibili in internet.

Le caratteristiche generali di Java

Java è stato creato dalla Sun Microsystem che ne ha rilasciato la prima versione nel 1995, successivamente nel 2010, la Sun è stata acquistata dalla Oracle.

Una caratteristica importante di Java è la capacità di costruire applicazioni che sono portabili su differenti piattaforme.

La portabilità è la capacità di un programma di essere eseguito su piattaforme diverse senza dover essere modificato e ricompilato. Le piattaforme sono diverse se sono basate su un diverso processore oppure su un diverso sistema operativo. Tra le piattaforme più usate ci sono Windows, Linux e Mac OS X per computer Apple.

Le caratteristiche generali di Java

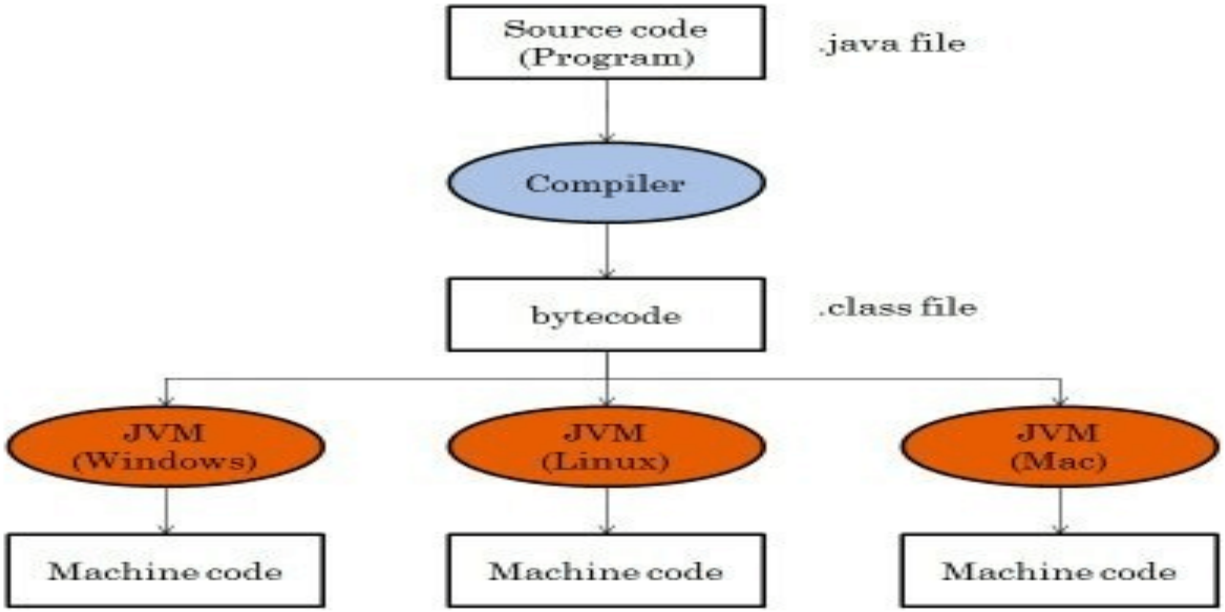
Quando si compila il codice sorgente scritto in Java, il compilatore genera il codice compilato chiamato bytecode. Questo è un formato intermedio indipendente dall'architettura ed è usato per trasportare il codice in modo efficiente tra varie piattaforme hardware e software.

Il bytecode non può essere eseguito da una macchina reale perché è stato progettato per essere eseguito da macchina astratta, detta Java Virtual Machine (JVM).

La JVM è il modulo di Java che crea l'ambiente di runtime per l'esecuzione del codice compilato e traduce ogni istruzione del bytecode nella corrispettiva istruzione della macchina reale su cui si sta eseguendo l'applicazione in quel momento.

Le caratteristiche generali di Java

Ogni piattaforma ha una sua JVM che interpreta il codice compilato e lo esegue sulla specifica macchina reale. Per questi motivi si può dire che java è un linguaggio interpretato, anche se la produzione del bytecode è effettuata con un'operazione di compilazione.



L’ambiente di sviluppo NetBeans

Quali strumenti avere?

Per scrivere applicazioni Java bisogna utilizzare un adeguato ambiente di sviluppo: java SDK (Java Software Development Kit), o più semplicemente JDK, è disponibile gratuitamente presso il sito web di Oracle (www.oracle.com).

Esistono numerosi ambienti di sviluppo IDE (Integrated Development Environment) attraverso i quali è possibile creare in breve tempo potenti applicazioni. Nel nostro corso usiamo NetBeans, intuitivo e potente, anch’esso gratuitamente disponibile.

Installazione di JDK e NetBeans

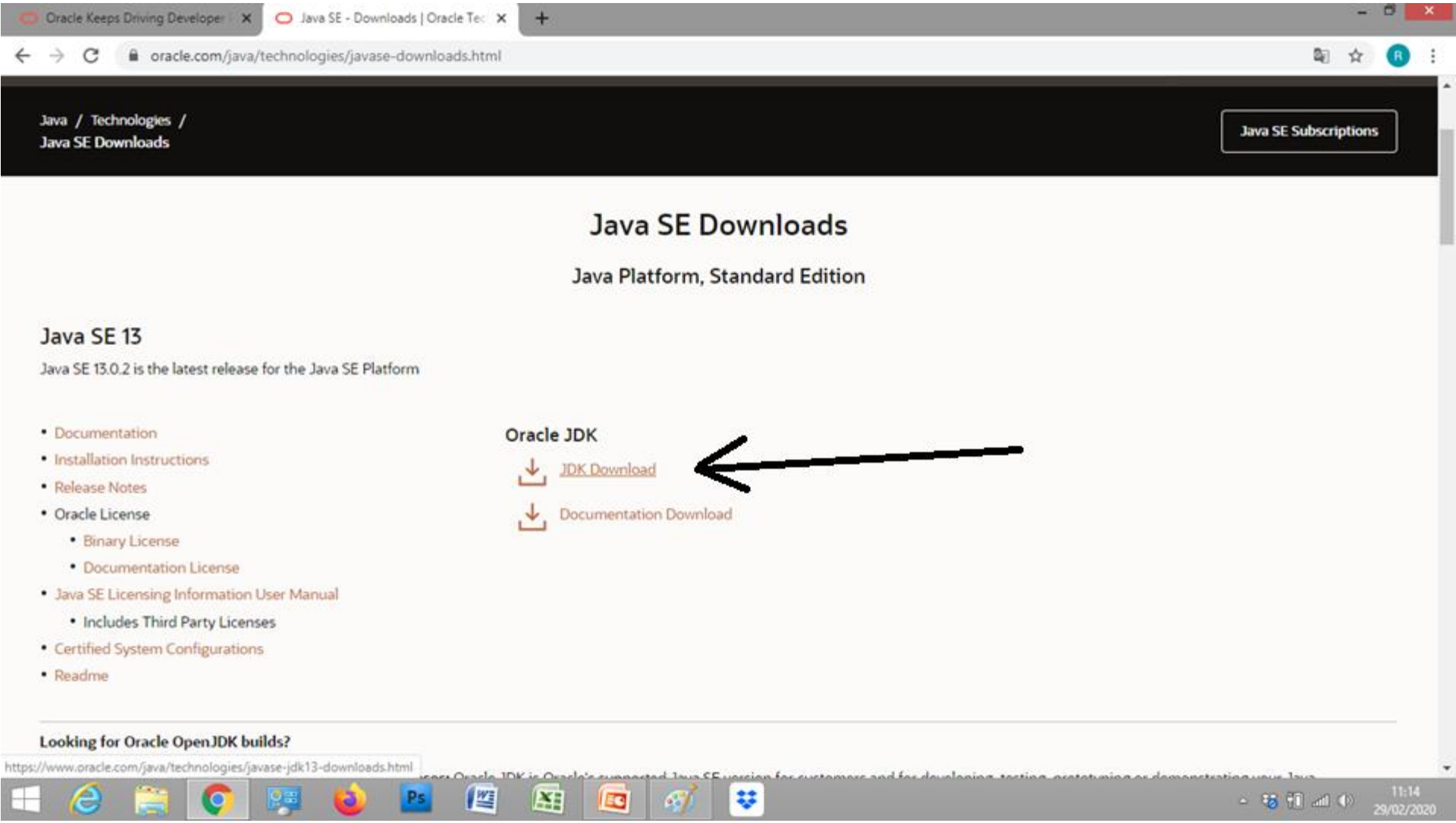
Per avere a disposizione un ambiente all'interno del quale realizzare i nostri programmi in java abbiamo bisogno di scaricare e installare i seguenti software:

- **JDK;**
- **L'ambiente IDE scelto (nel nostro caso NetBeans).**

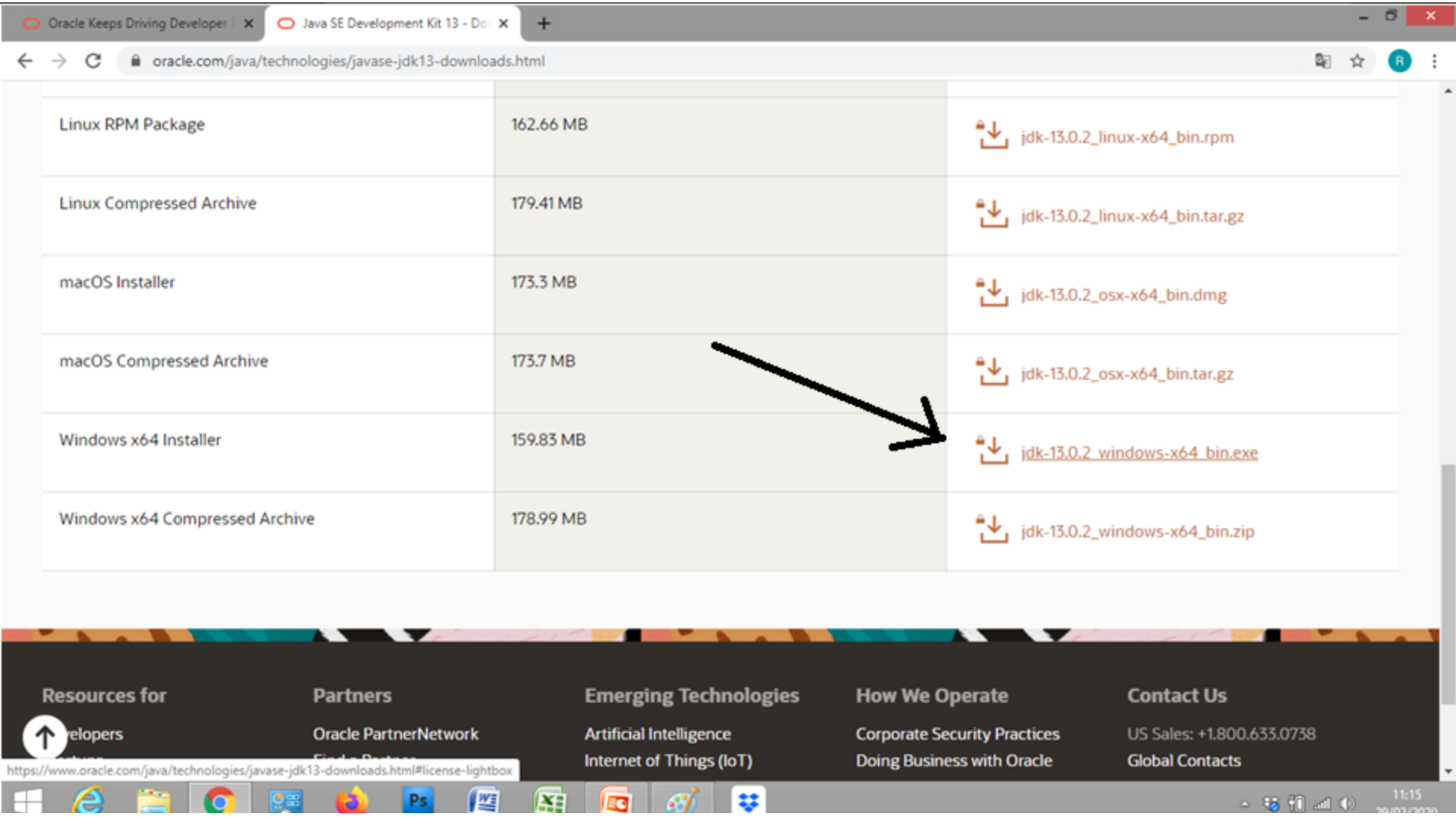
Per quanto concerne il primo, il sito di riferimento è quello della Oracle:

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

Installazione di JDK e NetBeans

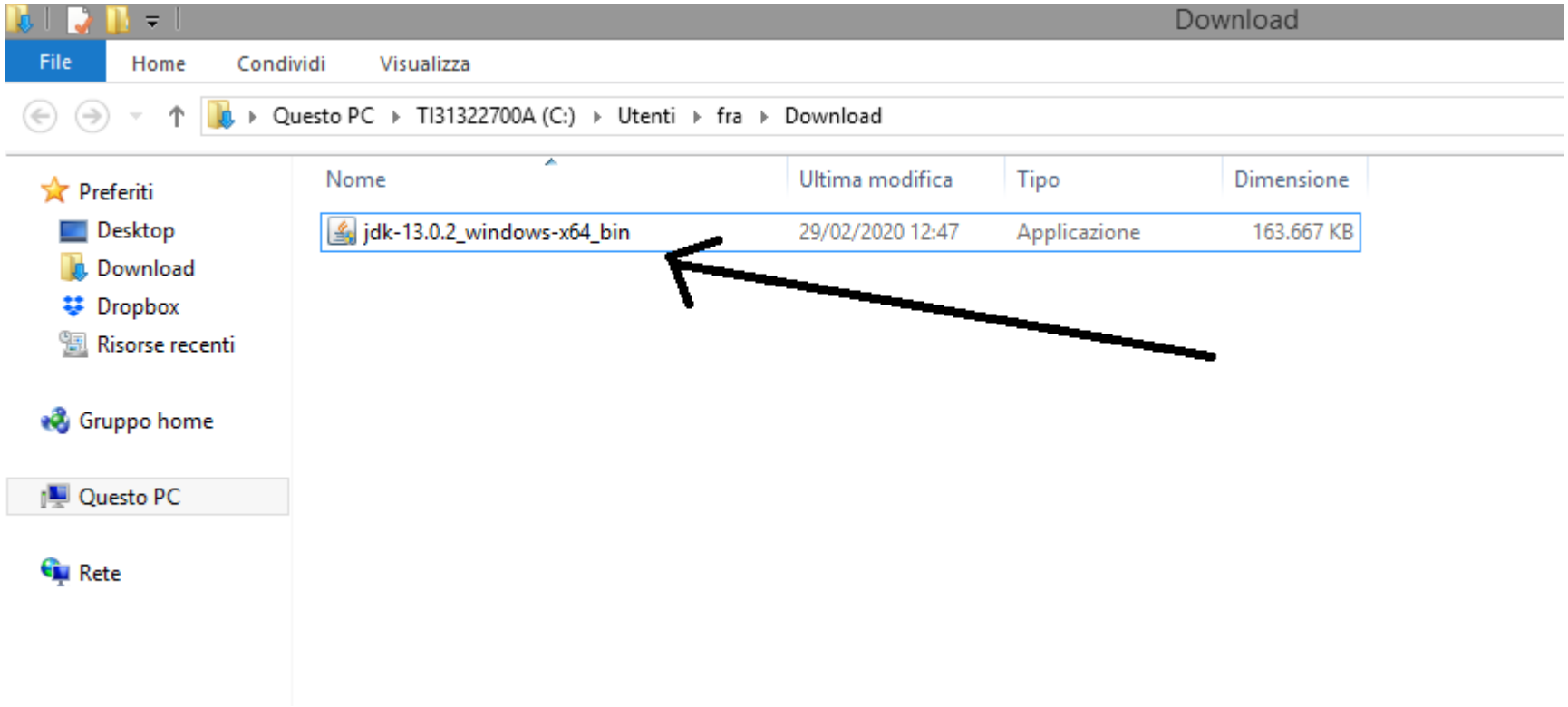


Installazione di JDK e NetBeans



Installazione di JDK e NetBeans

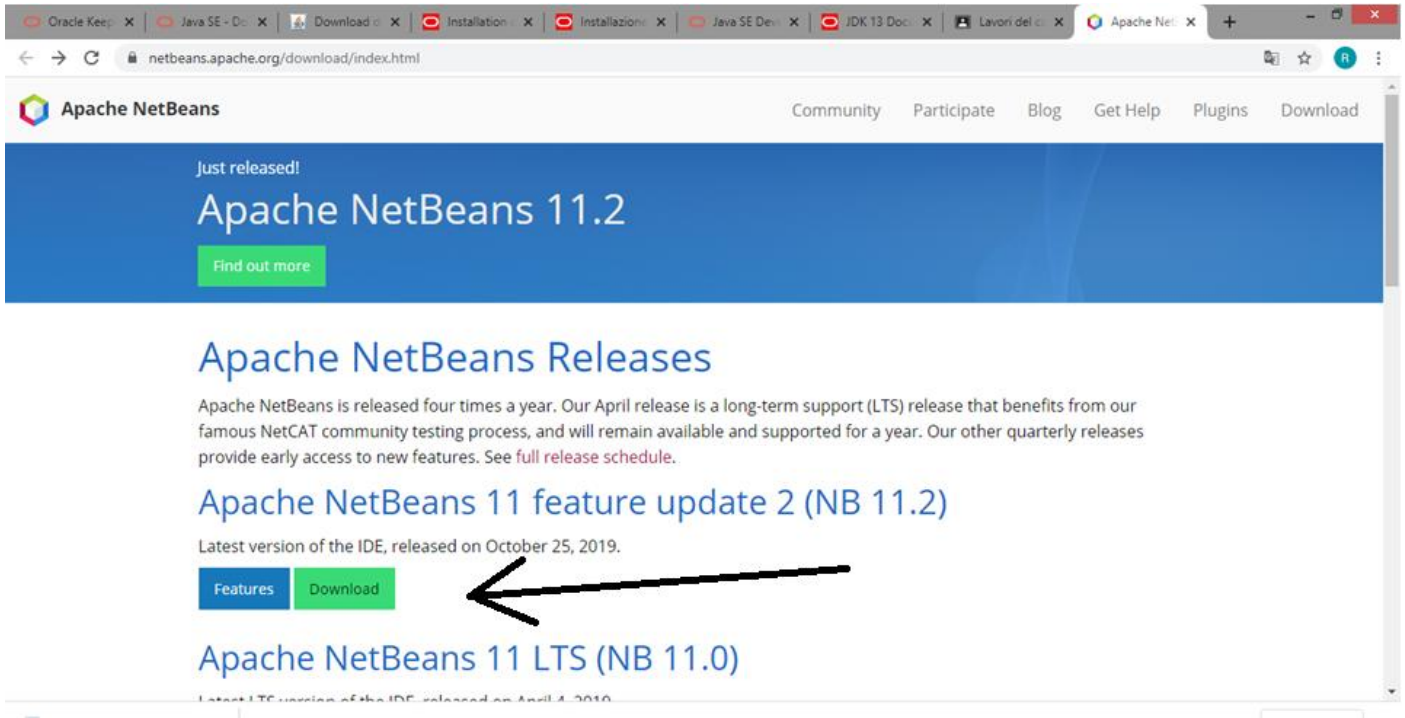
Quindi, una volta scaricato il software, posizionarsi nella cartella download ed installare il pacchetto.



Installazione di NetBeans

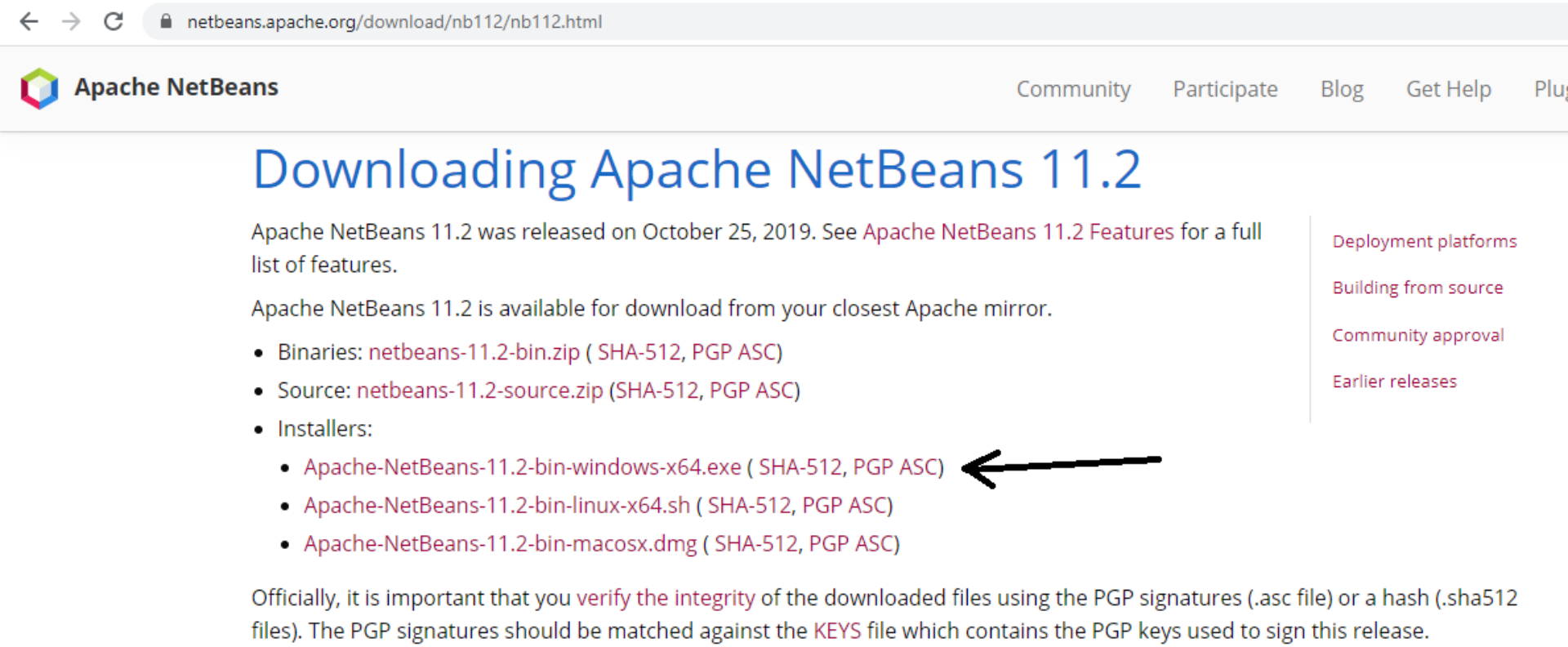
L'IDE NetBeans è open source: scritto in Java e implementato dal team di Oracle, ora è diventato un progetto di Apache incubator, cambiando nome in Apache Net-Beands. Pertanto è reperibile all'indirizzo

https://netbeans.apache.org/download/index.html



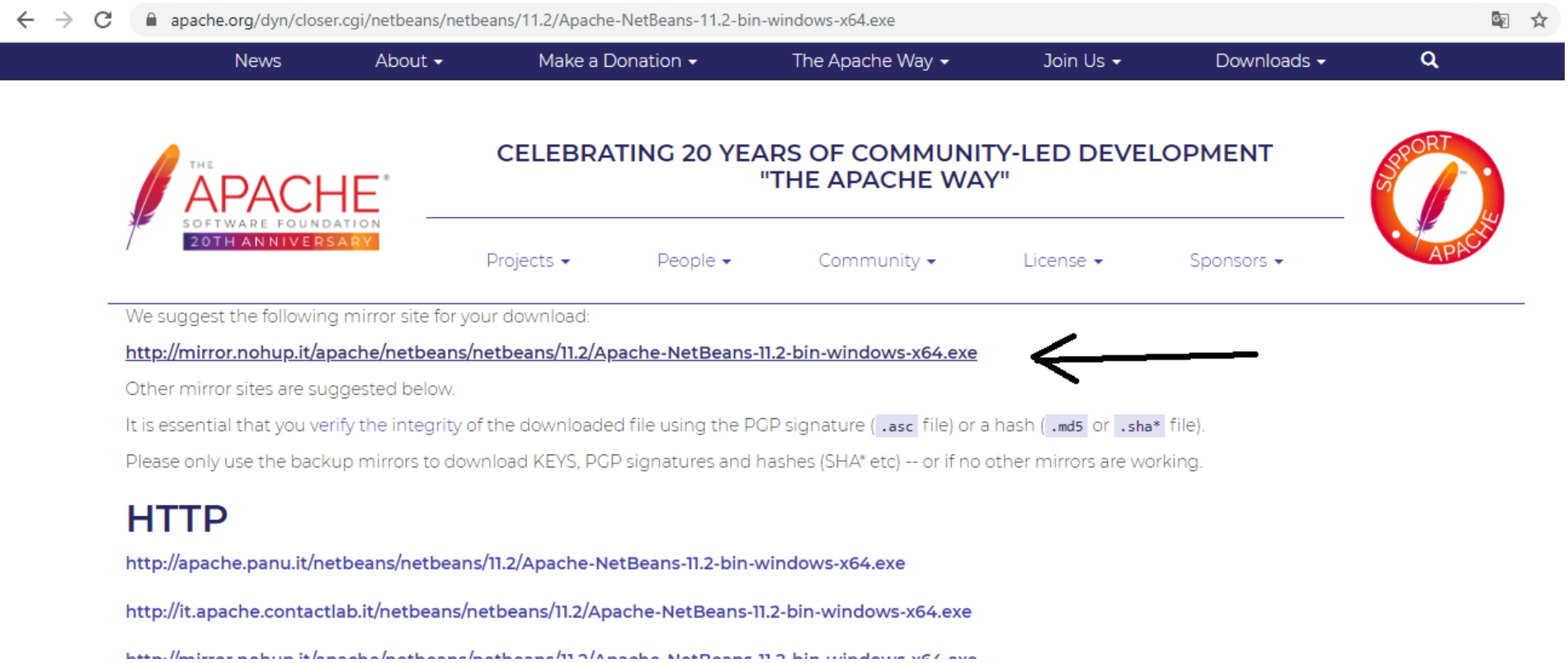
Installazione di NetBeans 11.2

L'ultima versione disponibile è al momento la 11.2. che dispone di un installer:



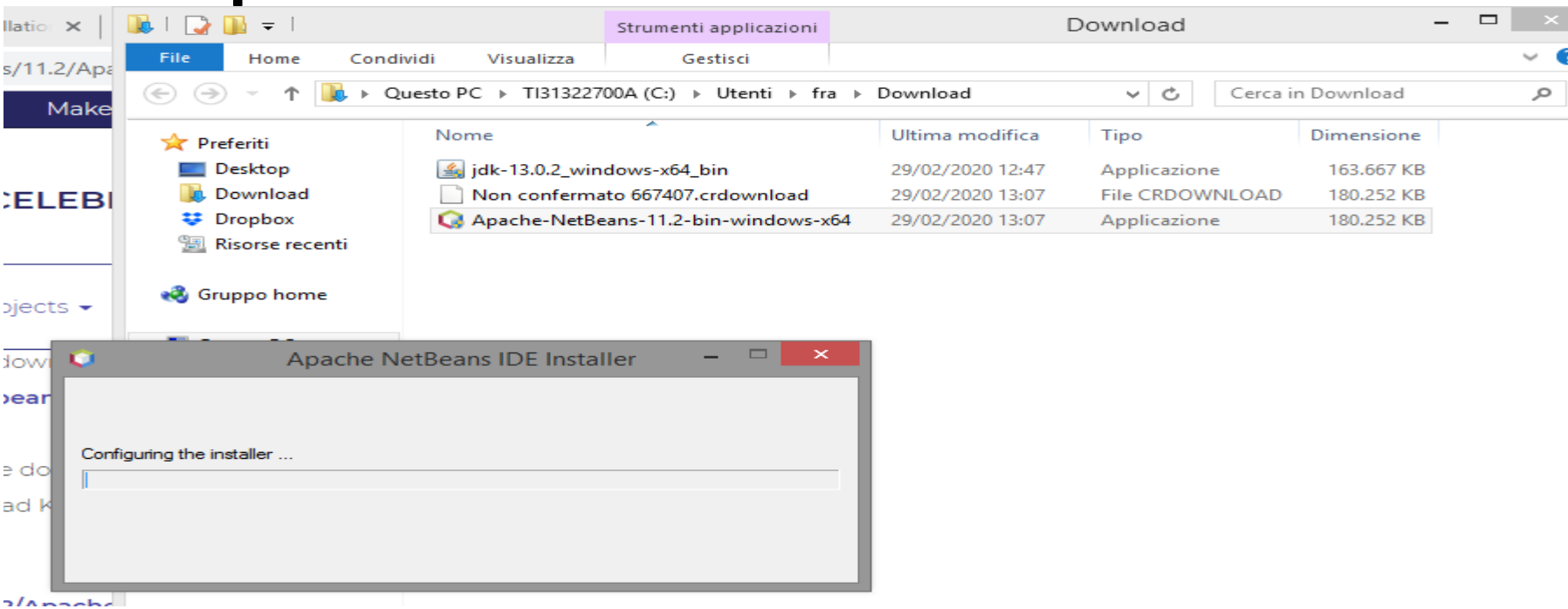
Installazione di NetBeans 11.2

Scaricare il pacchetto ed installare il tutto.



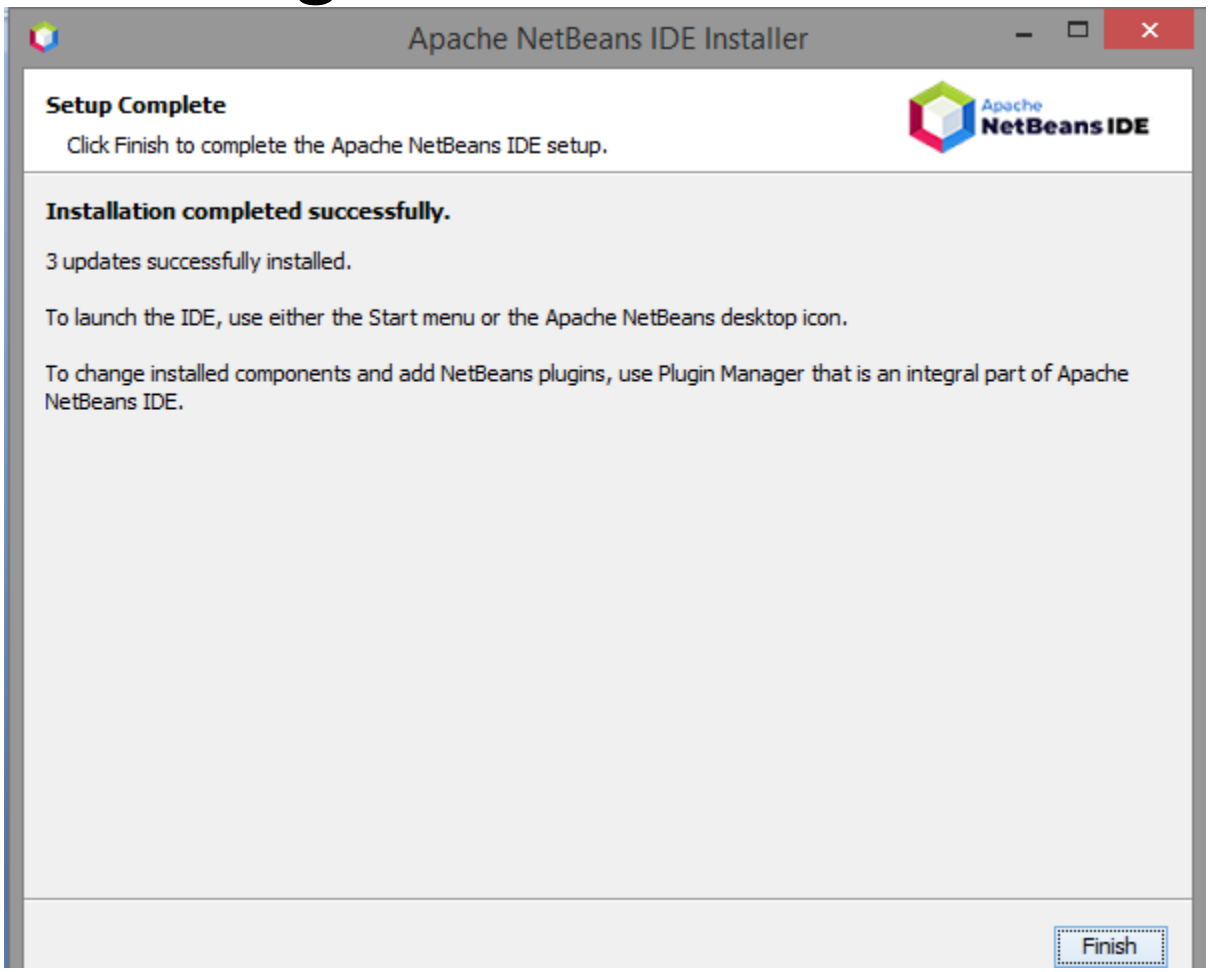
Installazione di NetBeans 11.2

Sempre nella cartella download, lanciare l'installazione di Netbeans. Si ricorda che se la versione JDK è incompatibile con la versione di NetBeans installata, allora scaricare e installare il JDK compatibile.



Installazione di NetBeans 11.2

Al termine dell'installazione verrà visualizzata la seguente immagine.



Installazione alternativa di NetBeans - older releases

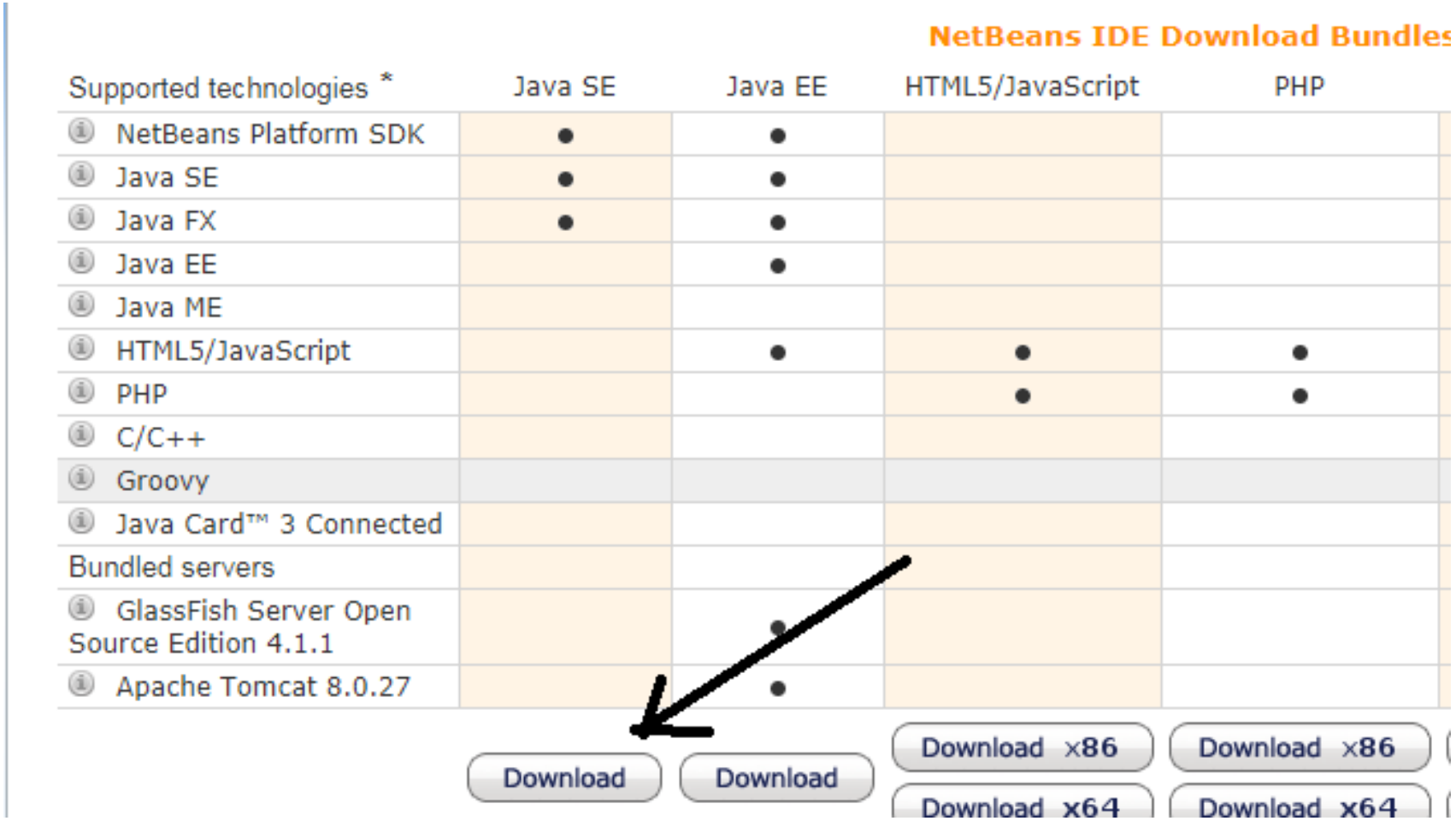
In alternativa, qualora si dovesse riscontrare l'incompatibilità tra la versione NetBeans 11.2 ed il proprio PC (tipico caso di coloro che hanno un sistema operativo Windows a 32 bit), sarà possibile installare una versione vecchia di NetBeans.

Per installare una versione meno recente, cliccare sul seguente link https://netbeans.apache.org/download/archive/index.html

Nel nostro caso, installiamo la versione NetBeans 8.2. (vedi fig. sotto).

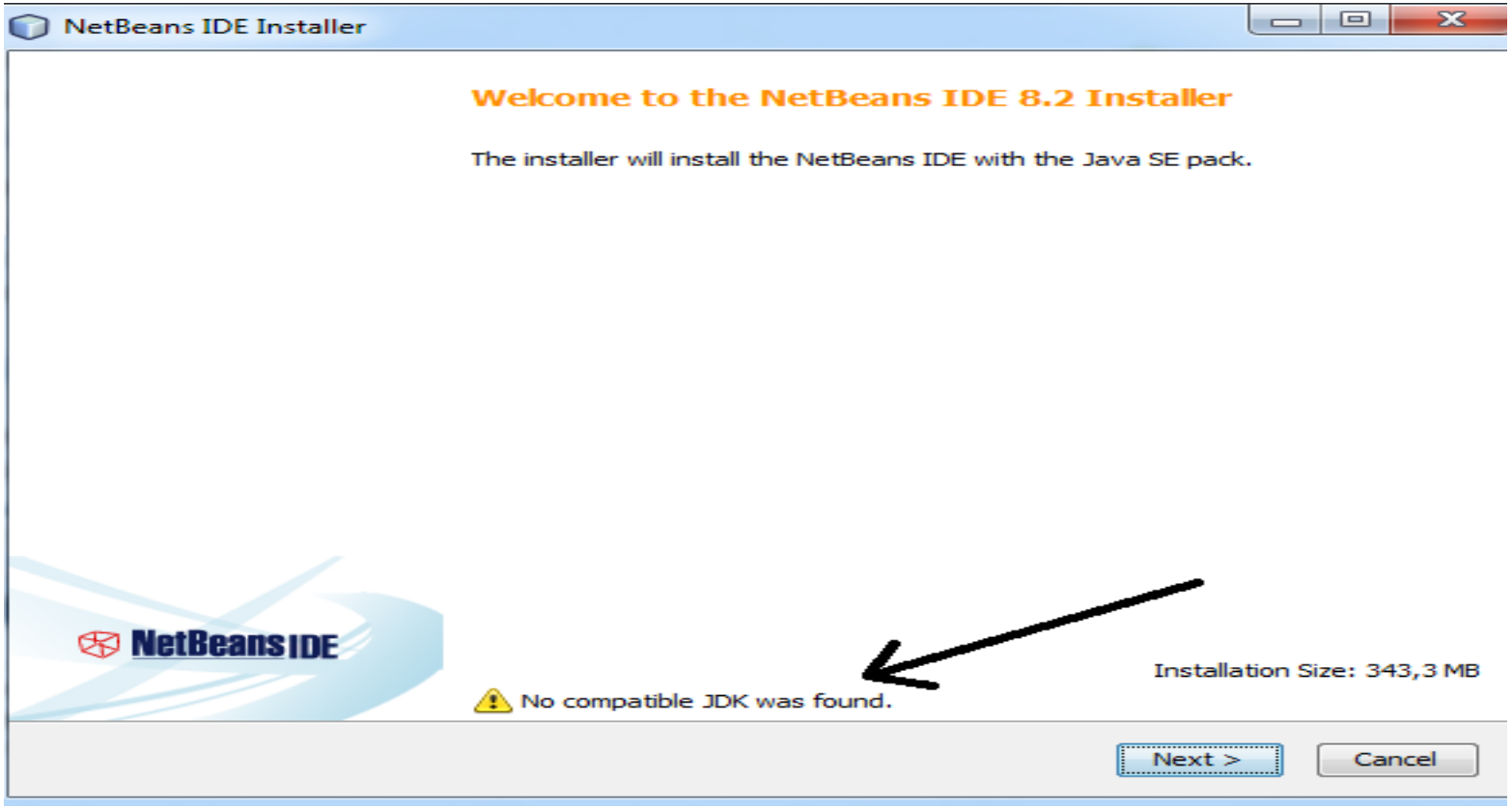
Installazione alternativa di NetBeans 8.2

Quindi, andare su Download per scaricare NetBeans 8.2, (Java SE)



Installazione alternativa di NetBeans 8.2

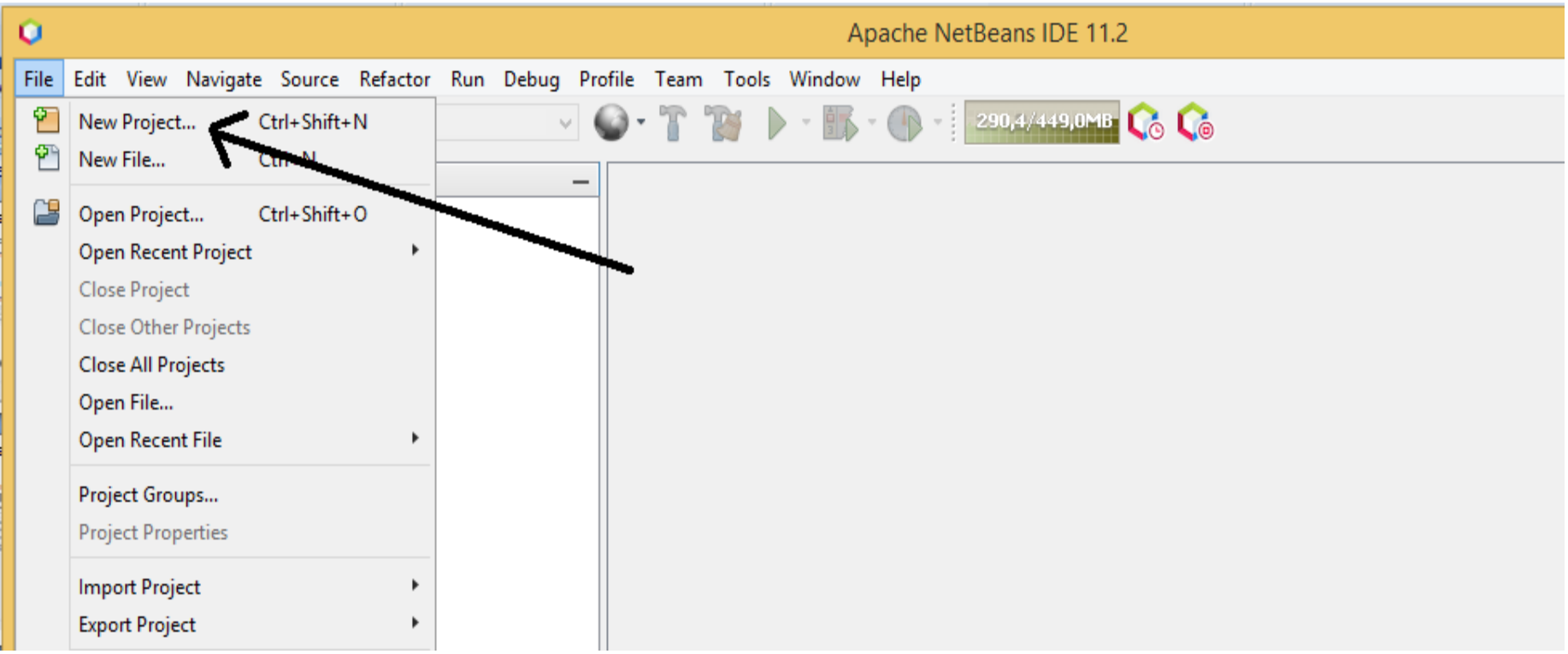
Un altro tipico errore che si può verificare durante l'installazione di Neteans 8.2 è la mancanza del pacchetto JDK (vedi figura sotto). In tal caso, bisogna uscire dall'installazione di NetBeans ed installare JDK 8 a 32 bit o versioni similari. Successiva ripetere l'installazione di NeteBans 8.2.



Avvio e funzionalità di NetBeans 11.2

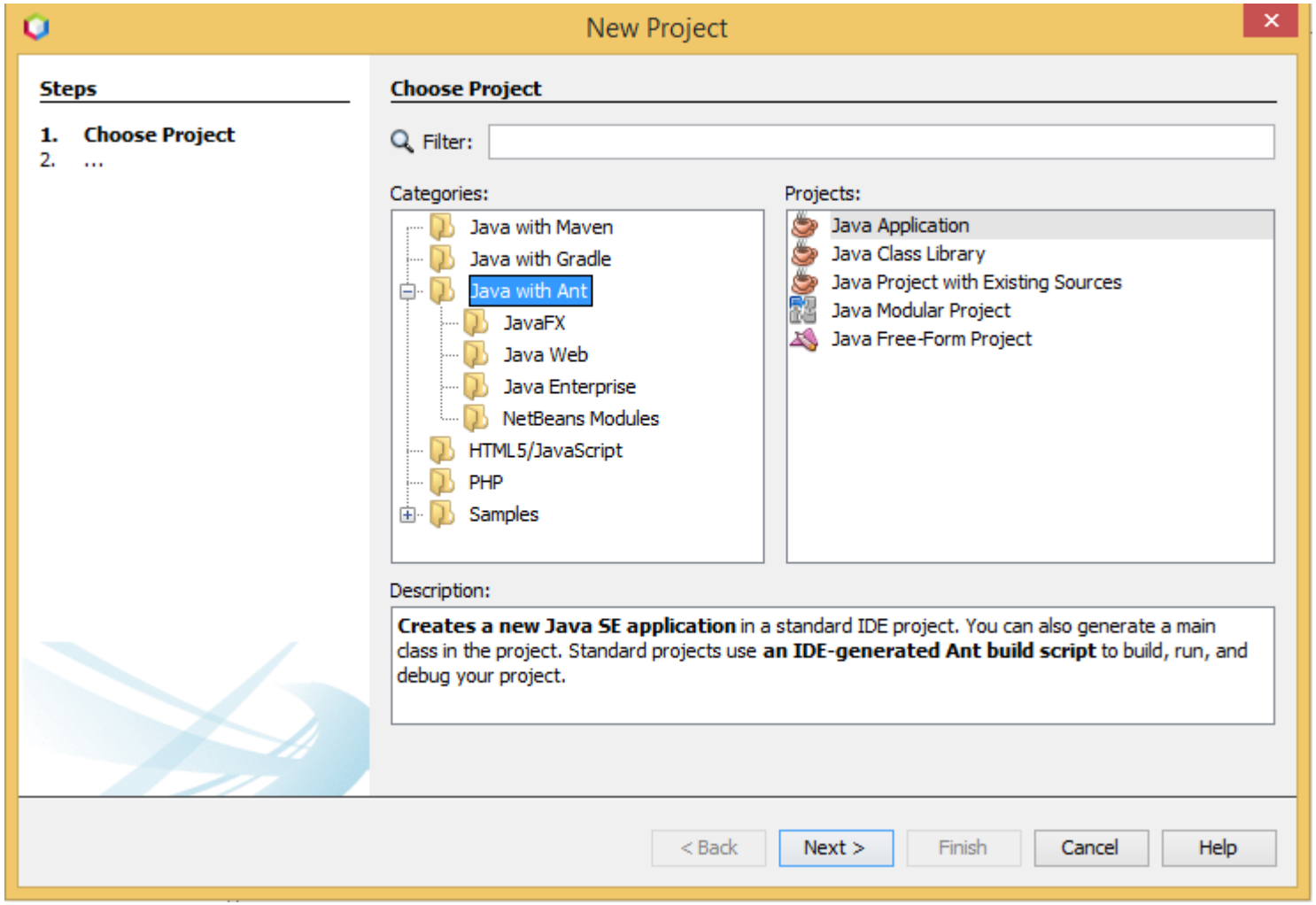
Per iniziare a utilizzare la piattaforma consigliamo di creare un nuovo progetto Java Application: pertanto, dopo l'avvio di NetBeans, bisogna compiere i seguenti passi.

Andare su File->New Project



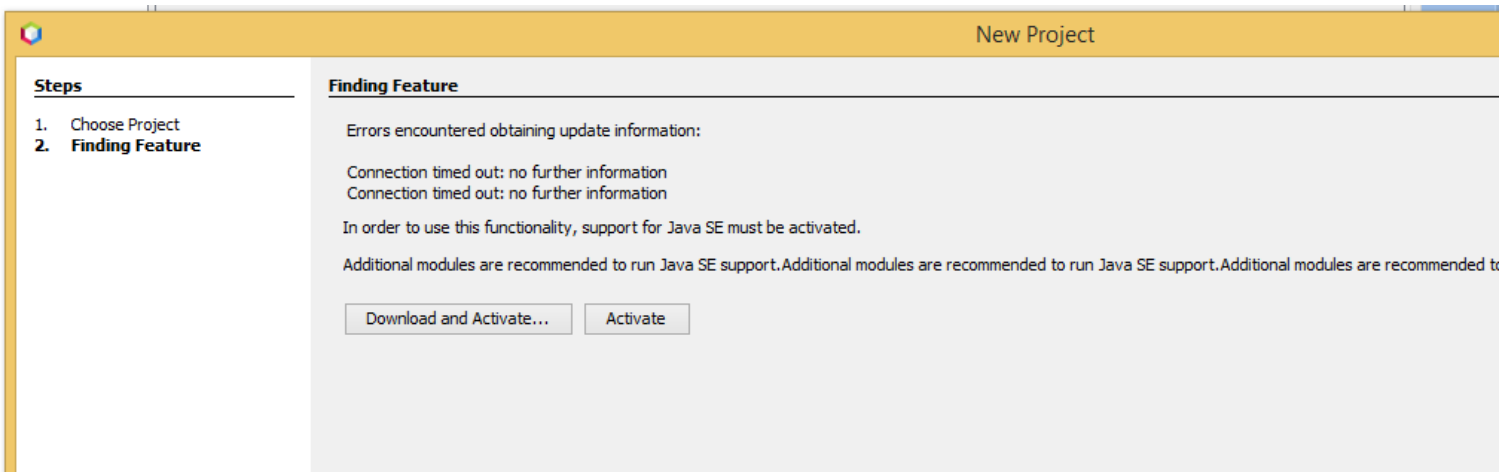
Avvio e funzionalità di NetBeans

Quindi selezionare Java with Ant e Java Application.



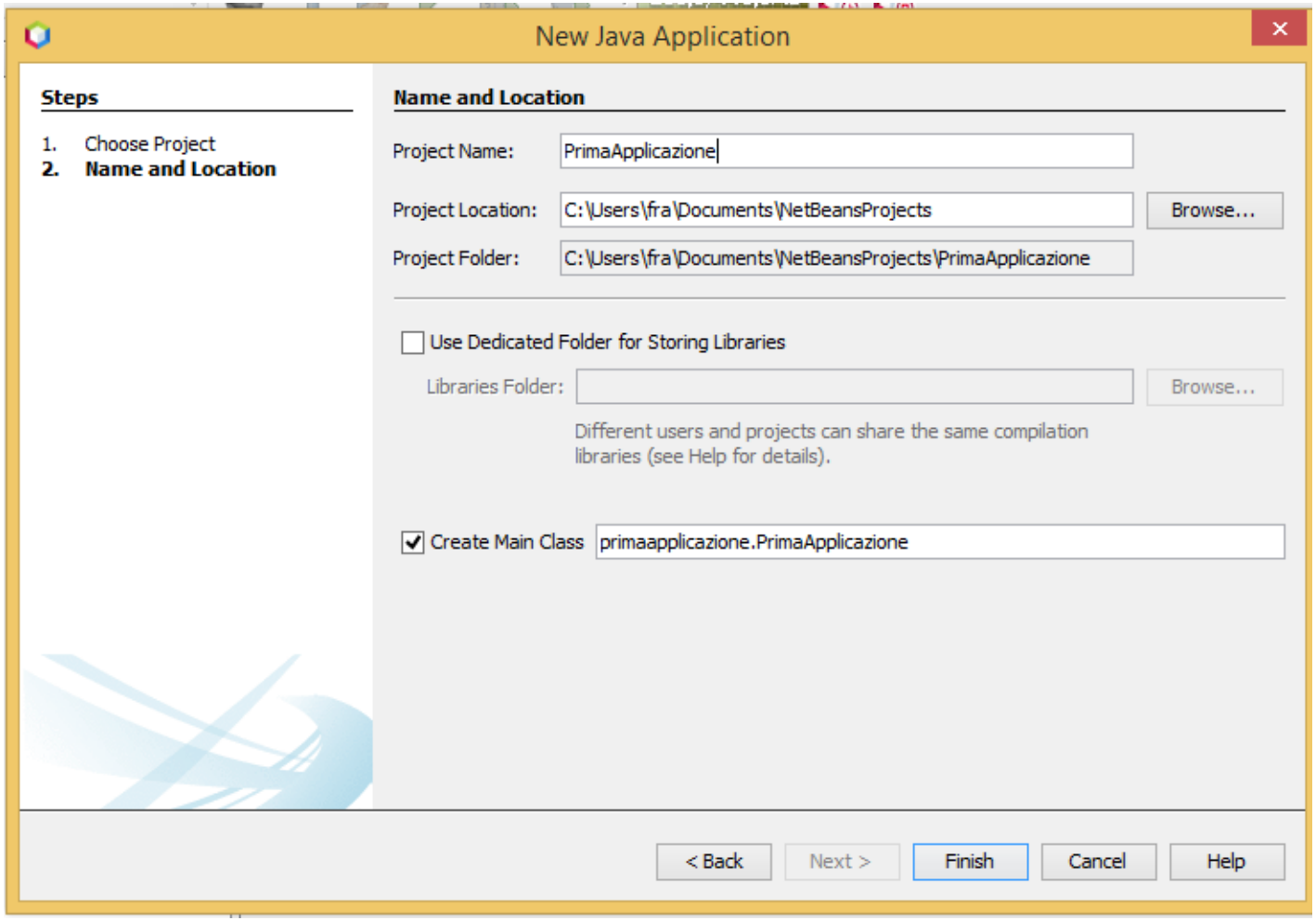
Avvio e funzionalità di NetBeans

La seguente schermata consente di scaricare e attivare dei moduli addizionali necessari all'utilizzo dell'applicazione.



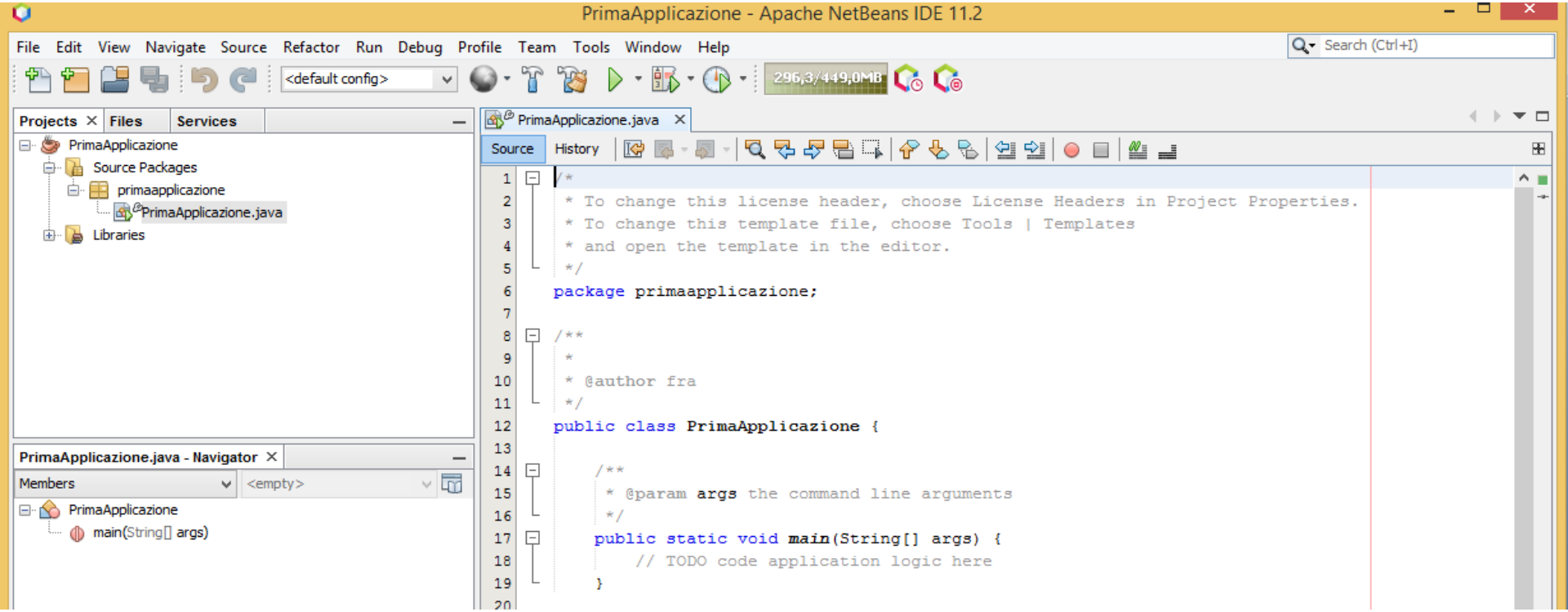
Avvio e funzionalità di NetBeans

Inserire il nome della tua prima applicazione (Project Name) ed eventualmente un percorso diverso da quello proposto.



Avvio e funzionalità di NetBeans

Il clic sul pulsante Finish visualizzerà, all'interno della piattaforma, una scheda che conterrà il codice sorgente di Java (con estensione .java) con alcune linee di codice già inserite. Nella parte sinistra, invece, verranno visualizzate le risorse in termini di file e componenti del progetto.



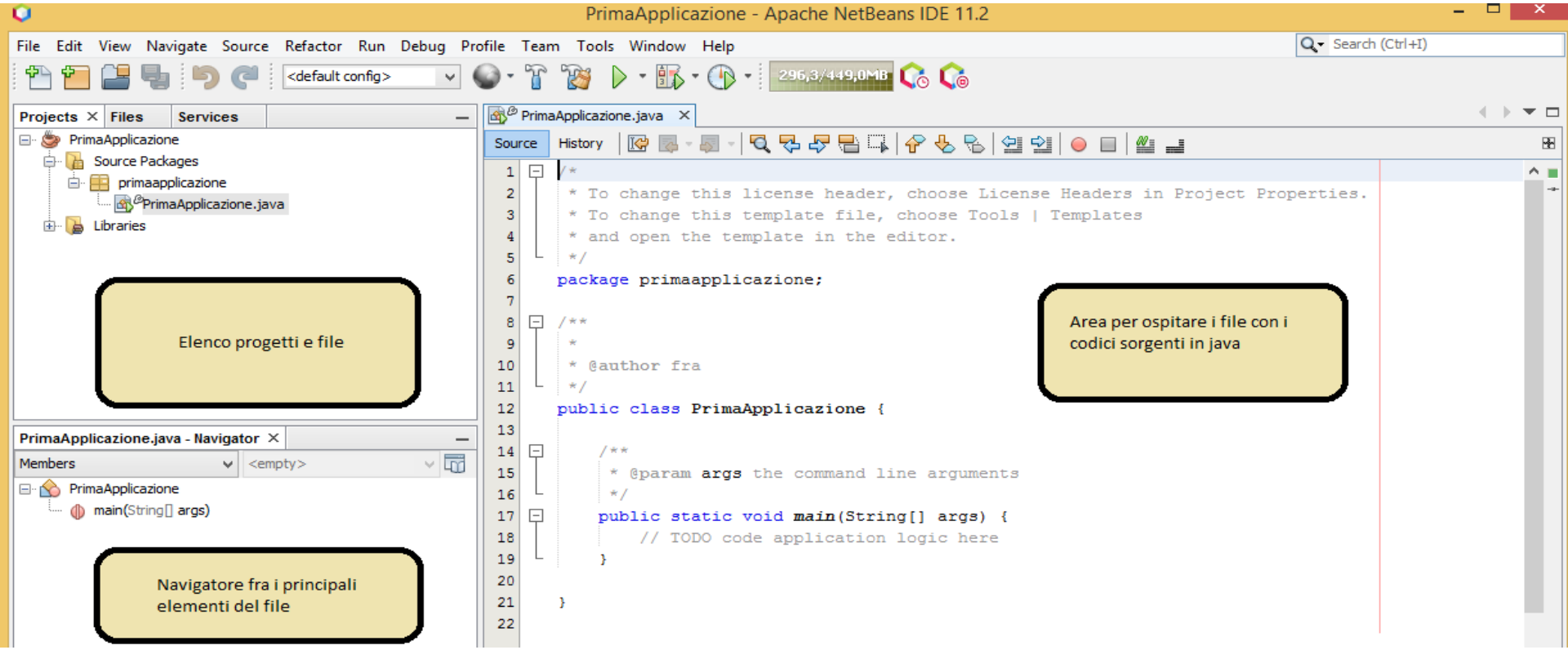
Avvio e funzionalità di NetBeans

Indichiamo di seguito alcune aree secondo la propria funzione:

-Area elenco Progetti e File.

-Area Navigatore fra i principali elementi del file.

-Area per ospitare i file con i codici sorgenti di java.



Avvio e funzionalità di NetBeans

Le similitudini fra Java e C++

Il linguaggio Java, a parte qualche piccola differenza, è identico al C++ nei seguenti aspetti e sintassi:

- **gestione dei commenti;**
- **concetto di blocco fra parentesi graffe;**
- **case sensitive;**
- **dichiarazione di variabili;**
- **operatori matematici binari e unari e ternario;**
- **operatori di confronto;**
- **operatori di assegnamento;**
- **espressioni aritmetiche;**
- **istruzioni condizionali (if…else, switch…case);**
- **istruzioni iterative (while, do …while, for).**

A differenza del C, Java è nato esclusivamente come linguaggio a oggetti. Prima di mettere mani all'interno del linguaggio Java è quindi doveroso fare una premessa sul concetto di programmazione a oggetti.

Una brevissima anticipazione sugli oggetti

Il paradigma della programmazione orientata agli oggetti (**OOP, Object-Oriented-Programming)** ha come concetto chiave quello di **oggetto**, che è una metafora del concetto di oggetto del mondo reale. In un linguaggio di programmazione orientato agli oggetti non si definiscono i singoli oggetti, ma si determinano le caratteristiche generiche che ognuno di essi deve possedere per poter far parte di una famiglia di oggetti, detta **classe.**

Se consideriamo, ad esempio, la classe persona, possiamo dire che essa descrive le caratteristiche comuni che i singoli oggetti "persona" devono possedere per farne parte, e dunque qualcosa come: "ha occhi", "ha capelli", "ha braccia", "ha gambe",ecc.....e le azioni che la persona può compiere, per esempio:"camminare", "prendere","correre", ecc.

La classe persona fa quindi da modello astratto del concetto di persona; **"modello astratto"** vuol dire che descrive soltanto le caratteristiche di una persona, ma ancora non è una persona concreta: affinché si abbia qualcosa di concreto con cui sia possibile interagire occorre creare una **istanza** della classe persona, ovvero un particolare oggetto appartenente a quella classe. Riepilogando:

-una **classe** è un modello astratto, generico, per una famiglia di oggetti con caratteristiche simili;

-una **istanza** di una classe, od **oggetto**, è una rappresentante concreto e specifico di quella classe.

Una brevissima anticipazione sugli oggetti

Ad esempio:

•**Classe**:automobili (hanno tutte una targa, un colore, un modello ecc);

•**Istanza** (oggetto): Fiat Panda Bianca (un particolare esemplare della classe automobili: con una sua targa, quel colore, ecc).

Un'applicazione Java qualsiasi può fare riferimento a una o più classi.

Una classe contiene **attributi** e **metodi**.

Gli **attributi**, o proprietà, specificano le caratteristiche degli elementi della classe, determinandone l'aspetto, lo stato, le altre qualità.

Esempi di attributi possono essere la targa, il colore, il modello di un'automobile. E' importante pensare agli attributi come gli ***aggettivi*** che descrivono gli oggetti.

In Java, per far riferimento alla proprietà di un oggetto useremo la seguente sintassi:

<NomeOggetto>.<Proprietà>; (si noti che il nome oggetto e la proprietà è separata da un **punto:dot notation**)

I **metodi** specificano la funzionalità che la classe offre, cioè le operazioni che un oggetto appartenente a quella classe è in grado di compiere e che corrispondono ai suoi comportamenti. Esempio di metodi possono essere "si avvia", "gira a destra", ecc.

E' utile pensare ai metodi come ai ***verbi*** che esprimono l'azione da compiere.

Anche per i metodi seguiremo la stessa sintassi vista per le proprietà:

<NomeOggetto>.<Metodo>([ListaParametri]); (anche qui c'è il punto che separa)

La programmazione in Java non può prescindere dal concetto di classe anche quando non vi è interazione con gli oggetti.

La struttura di base di un'applicazione Java

Un'applicazione in Java è costituita da uno o più file con estensione .java, ognuno dei quali deve contenere una classe che dà il nome al file stesso. Una sola classe, definita principale, può contenere il metodo main(), da cui comincia l'esecuzione dell'applicazione. La classe principale contiene dichiarazioni di variabili e istruzioni che compongono la struttura principale del flusso di esecuzione del programma. Riprendiamo il progetto aperto con NetBeans precedentemente ed aggiungiamo nell'area che ospita il file sorgente di Java, nel metodo main(), la prima istruzione che consente di stampare una stringa:

System.out.println("prima applicazione Java");

System.out rappresenta un **oggetto** che visualizza un messaggio sul dispositivo di output standard del sistema, il monitor.

In particolare:

-**System** è il nome della ***classe*** che contiene oggetti e metodi per accedere alle risorse del sistema e per descriverne i comportamenti specifici;

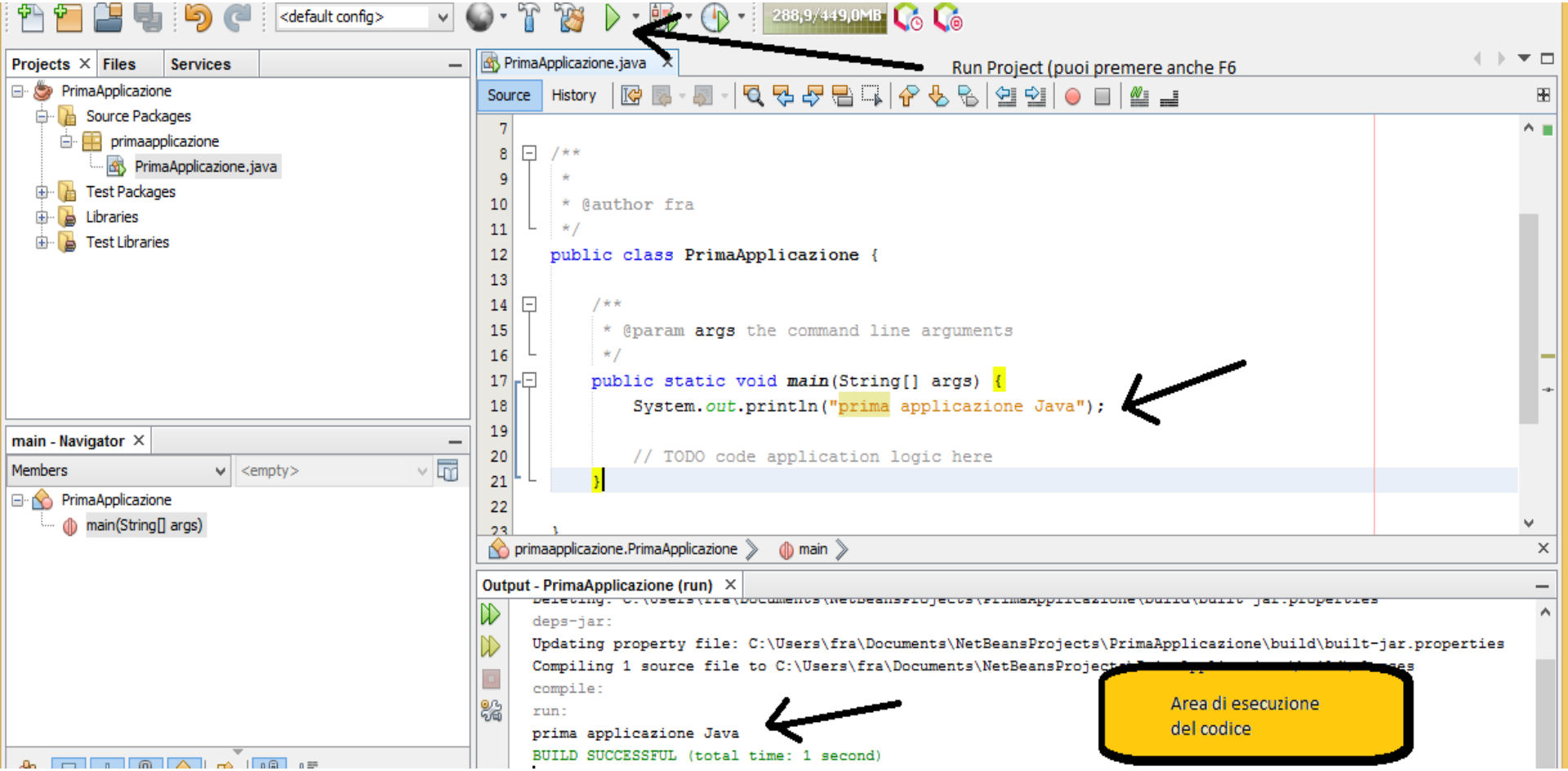
-**out** è uno ***stream o flusso*** predefinito;

-**print()** è il ***metodo*** che ha come argomento il valore da visualizzare; tale argomento è di tipo string, ovvero è una stringa (insieme di caratteri). Nell'esempio sopra abbiamo usato il metodo println, ossia stampa e va a capo.

Convenzionalmente, i nomi delle classi java hanno l'iniziale maiuscola, mentre i nomi degli oggetti e dei metodi iniziano con la minuscola.

La struttura di base di un’applicazione Java

Mandiamo in esecuzione il nostro primo programma, cliccando sulla freccia verde e dopo un po di tempo, se è tutto corretto, si vedrà nell’’area di esecuzione la stampa della stringa “prima applicazione in Java”. Infine salviamo il progetto.



La struttura di base di un'applicazione Java

La prima istruzione che troviamo nel file è:

package PrimaApplicazione;

Riporta automaticamente il nome del progetto (tutto scritto in minuscolo) in tutti i file che appartengono allo stesso progetto. Un Package costituisce pertanto una libreria formata da differenti classi e quindi da diversi file.

L'istruzione:

public class PrimaApplicazione

Determina il nome della classe e la visibilità della stessa (public), che consente alla classe l'eventuale interazione con le altre.

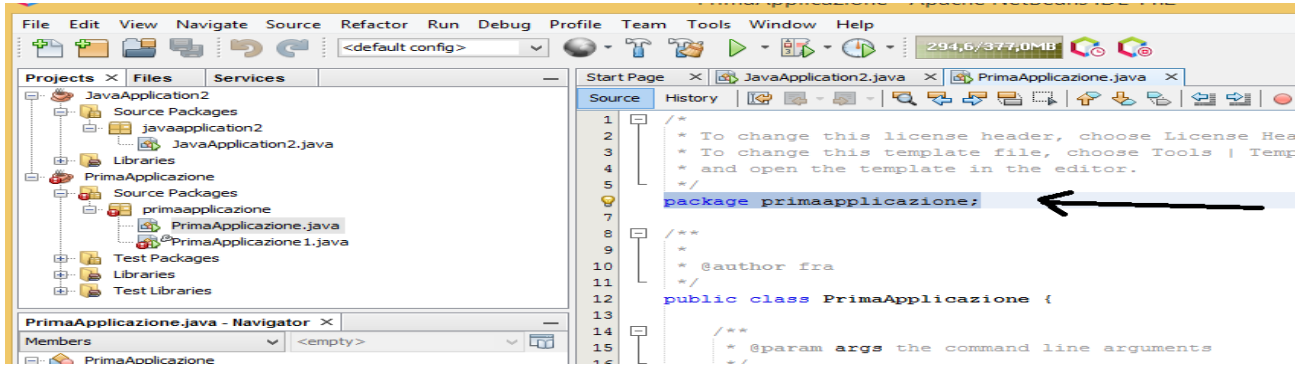
Il metodo **main(),** è un particolare metodo che governa il flusso di esecuzione dell'intero programma. In quanto metodo, non può esistere autonomamente, ma deve essere necessariamente inserito in una classe.

Il metodo main() può essere preceduto da alcune parole chiave:

-**public**: indica che il metodo è pubblico, cioè visibile da qualsiasi classe del programma;

-**static**: indica che il metodo è associato alla classe ed esiste indipendentemente da istanze della classe;

-**void**: indica che il metodo non restituisce valori di ritorno.



Variabili e tipi primitivi in Java

Una **variabile** è un contenitore di dati situato in una porzione di memoria e destinata a contenere valori. Una variabile è caratterizzata dal fatto che il dato memorizzato in essa può con il tempo essere modificato (cambiare).

Una variabile per essere utilizzata deve essere prima dichiarata. Per dichiarare una variabile, dobbiamo definire il tipo di dato da contenere ed il nome con il quale individuare la variabile. Per esempio, se vogliamo creare una variabile di nome somma che dovrà contenere solo numeri interi, si potrà scrivere la seguente istruzione in Java: int somma;

il nome di una variabile è inteso solitamente come una sequenza di caratteri seguiti anche da cifre (esempio: somma12). Il tipo di dati primitivi utilizzabili in Java sono:

Tipo	Descrizione	Spazio in memoria	Range di variabilità
Int	Intero normale	32 bit	da -2.147.483.648 a +2.147.483.647
short	intero corto	16 bit	da -32.768 a +32767
long	intero lungo	64 bit	da -9.223.372.036.854.775.808 a +9.223.372.036.854.775.807
byte	intero byte	8 bit	da -128 a +127
boolea	booleano	8 bit	true, "vero" - false, "falso"
char	carattere	16 bit	da 0 a 65536 senza segno
float	numero con la virgola mobile precisione singola	32 bit	valori negativi e positivi da 1,40129846432481707x10^-45 a 3,4282346638528860x10^38
double	numero in virgola mobile precisione doppia	64 bit	valori negativi e positivi da 4,94065655841246544x10^-234 a 1,79769313486231570x10^138

I tipi primitivi in Java

Quindi per dichiarare in Java una variabile di nome x e y di tipo intero, scriverò:

```
int x,y;
```

Per dichiarare variabili di altro tipo avrò:

```
double k;
float w;
```

E' possibile dichiarare ed inizializzare, ossia assegnare un valore alla variabile, direttamente con un'unica istruzione:

```
double numero=345.7;
```

I valori decimali con il simbolo "." (punto) saranno considerati di tipo double a meno che non sia posposta la lettera "F" o "f" (float).

La seguente assegnazione produrrà un errore:

```
float num=345.7;     errore!!!
```

Bisogna scrivere

```
float num=345.7f;
```

Le costanti

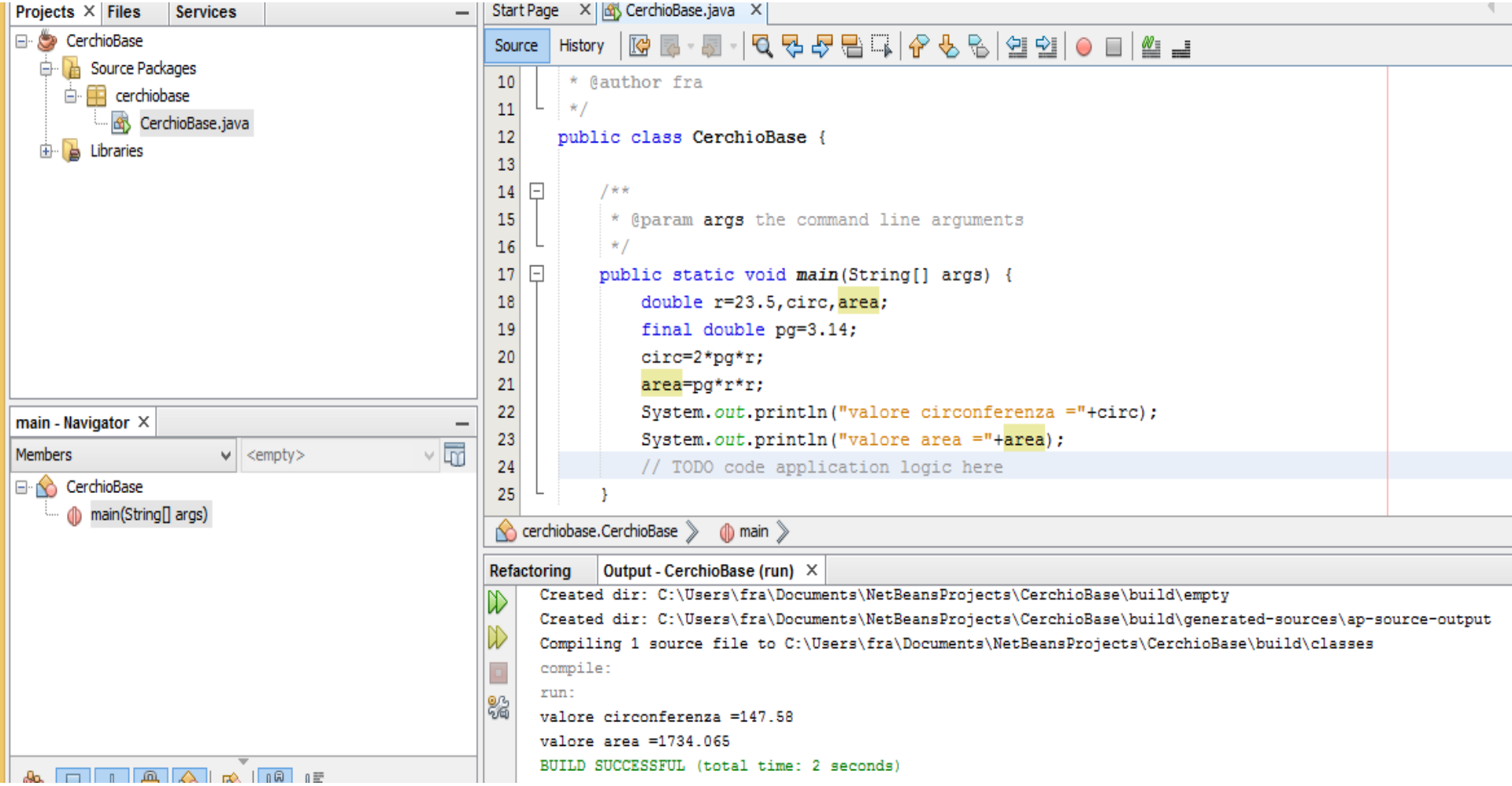
Le costanti seguono la stessa sintassi delle variabili alla quale va anteposta la parola **final** a testimonianza del fatto che il valore contenuto in esso non può essere modificato durante l'esecuzione del programma:

```
final int n=10;
```
costante di tipo intero

```
final char c='a';
```
costante di tipo carattere

Secondo programma in Java

Il programma successivo crea una classe in Java che stampa il valore della circonferenza e dell'area di un cerchio di raggio r. Nell'area del codice sorgente digitiamo il programma, al temine, dopo la compilazione, possiamo notare nell'area di esecuzione del codice (finestra in basso a destra) i valori della circonferenza e dell'area stampati.



La visibilità delle variabili

Non esiste un punto preciso del programma dove dichiarare le variabili. Per comodità e leggibilità del codice conviene raggruppare le dichiarazioni di tutte le variabili nella parte iniziale del blocco di programma in cui verranno utilizzate.

In linea di principio ogni variabile può essere dichiarate in qualsiasi punto ma, a seconda della posizione, cambia la sua visibilità.

Il campo di **visibilità** di una variabile è costituito dal blocco dentro al quale è stata dichiarata. All'esterno di questo blocco non viene riconosciuta. Inoltre, dopo la dichiarazione di una variabile , non è possibile dichiarare altre variabili con lo stesso nome, anche all'interno di blocchi annidati. In pratica non si possono verificare situazioni come la seguente:

```
{
    int a;
    ….
    ….
    {
        int a;  //errore !!!
        ……..
     }
     …..
 }
```

Si noti anche la modalità di rientranza delle istruzione annidate (tra parentesi graffe), tale modalità si chiama **indentazione** ed offre una visione più chiara dei blocchi annidati nel programma sorgente.

I tipi di carattere

Il tipo di dato carattere è indicato, come già anticipato precedentemente, con la parola chiave char. E' definito usando 16 bit e la codifica Unicode. Java non utilizza la codifica ASCII per rappresentare i caratteri. Esempio: char Iniziale='N';

La **codifica Unicode** definisce un insieme di caratteri usando due byte.

I primi 128 caratteri dell'insieme Unicode sono equivalenti ai caratteri nel codice ASCII. Alcuni caratteri, che non possono essere digitati, vengono rappresentati attraverso la sequenza di escape. Queste sono formate da un backslash (\) seguito da una lettera o da un simbolo.

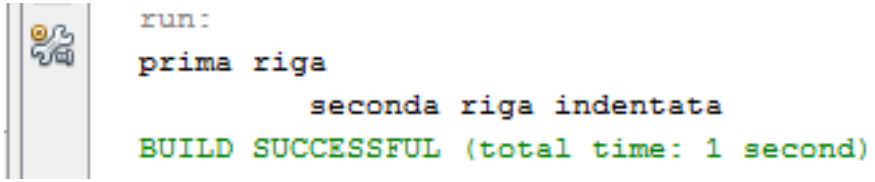
Alcuni esempi di sequenze di escape sono:

- **\n** ritorno a capo;
- **\t** tabulazione;
- **** backslash;
- **\"** doppio apice

Le stringhe sono un tipo di dato che raggruppa un insieme di caratteri. Le stringhe non sono tipo predefinito, ma sono definite da apposita classe (String). La sequenza di escape possono essere usate all'interno di una stringa per stampare dei caratteri speciali. L'istruzione sotto:

```
System.out.println("prima riga \n\t seconda riga indentata");
```

Produrrà come stampa due righe una indentata rispetto all'altra, poiché è presente nella stampa dopo il ritorno a capo anche la tabulazione, ossia lo spostamento a destra della stampa.



I commenti e la documentazione

All'interno del programma Java possono essere inserite frasi contenenti commenti o annotazioni del programmatore, con cui è possibile documentare il significato delle variabili o delle istruzioni utilizzate.

In Java ci sono tre forme per rappresentare i commenti.

-La prima forma (commento riga) utilizza i caratteri //. Tutto quello che viene scritto dopo questi caratteri e sulla stessa riga è considerato un commento. Si usa per scrivere commenti brevi.

```
Int eta; // dichiarazione di una variabile intera per contenere l'età della persona
```

-La seconda forma di commento è /*.....*/ . Tutto il testo compreso tra i caratteri di inizio /* e quelli di termine */ viene considerato un commento. Questa forma consente di dividere il commenti su più righe.

```
/* commento che occupa
più righe*/
```

-La terza forma è /**….*/. Funziona come il commento su più righe. Rappresenta un commento di documentazione e può essere usata prima di dichiarare una classe o un metodo. Questa particolare forma di commento è importante perché viene riconosciuta dagli strumenti per la generazione automatica della documentazione dei programmi.

```
/** Funzione per calcolare il MCD tra due numeri  */
public void calcolaMCD(int a, int b)
{
   //istruzione
}
```

Il casting per la conversione di tipo

Il linguaggio Java richiede molta attenzione nell'uso dei tipi di dato: il compilatore controlla che gli assegnamenti vengano effettuati tra variabili dello stesso tipo o di un tipo compatibile. Quasi tutti questi controlli di correttezza di tipo vengono eseguiti durante la fase di compilazione. Si dice anche che Java è fortemente **tipizzato**.

Se per esempio si vuole assegnare un valore intero a una variabile di tipo float, Java effettua automaticamente la conversione perché il tipo float è più grande del tipo intero. Questa conversione, chiamata anche **promozione,** è permessa perché solitamente non si ha perdita di dati.

```
int num;
float  convirgola;
num=32;
convirgola=num;
```

In questo caso la variabile di tipo float (convirgola) può ricevere un valore intero che viene automaticamente convertito in float. La conversione da float a intero, invece, non viene eseguita automaticamente.

Quando invece si tenta di effettuare assegnamenti tra variabili ed espressioni con tipi che sono in conflitto, viene generato un errore.

```
int num;
float  convirgola;
convirgola=32.34f;
num=convirgola;    // il programma genera un errore
```

Il casting per la conversione di tipo

Se si vuole eseguire l'assegnamento visto nella pagina precedente, il programmatore deve indicare esplicitamente la sua volontà, forzando la conversione.
Il **casting** è il meccanismo che consente al programmatore di indicare la conversione da un tipo di dato a un altro tipo.
Il casting si effettua anteponendo al valore da convertire, tra parentesi tonde, il tipo di dato in cui sarà convertito.
Nell'esempio precedente, il casting viene effettuato aggiungendo (int) prima della variabile convirgola.

```
float  convirgola;
convirgola=32.34f;
num=(int) convirgola;    /* Operazione di casting eseguita correttamente. Nella variabile num resta memorizzato solo il numero 32. */
```

Altro esempio di casting:

```
int numeratore=20;
int denominatore=6;
double  risultato;
risultato=(double) numeratore/denominatore;
```

Per ottenere il risultato corretto con le cifre decimali, si deve effettuare un casting del numeratore prima di fare la divisione.

Gli operatori

Gli operatori sono caratteri speciali che identificano particolari operazioni sui dati. Per esempio si possono usare gli operatori matematici tra i numeri oppure l'operatore di concatenazione tra le stringhe.

Operatore	Descrizione	Uso	Significato
+	Somma	op1+op2	Somma il \valore di op1 a quello di op2
-	Sottrazione	op1-op2	Sottrae al valore di op1 quello di op2
*	Moltiplicazione	op1*op2	Moltiplica il valore di op1 con quello di op2
/	Divisione	op1/op2	Divide il valore di op1 con quello di op2
%	Modulo	op1%op2	Calcola il resto della divisione tra il valore di op1 e quello di op2
-	Negazione aritmetica	-op	Trasforma il valore di op in positivo o negativo

Operatori di assegnamento shortcut		
Operatore	**Utilizzo**	**Equivalente a**
+=	sx +=dx	sx = sx + dx;
-=	sx -=dx	sx = sx - dx;
*=	sx *=dx	sx = sx * dx;
/=	sx /=dx	sx = sx / dx;
%=	sx %=dx	sx = sx % dx;
&=	sx &=dx	sx = sx & dx;
|=	sx |=dx	sx = sx | dx;
^=	sx ^=dx	sx = sx ^ dx;
<<=	sx <<=dx	sx = sx << dx;
>>=	sx >>=dx	sx = sx >> dx;
>>>=	sx >>>=dx	sx = sx >>> dx;

Gli operatori di concatenazione

Le stringhe possono essere concatenate tra loro usando l'operatore di concatenazione +.
Esso consente anche di concatenare una stringa con un numero, eseguendo automaticamente la conversione del numero in stringa.
Questa possibilità è molto utile con il metodo println per l'output dei risultati, perché permette di stampare sia testo che numeri.

```
int sco=20;
System.out.println("sconto ="+sco+"%");
```

La notazione prefissata e postfissata.

Gli operatori di incremento e di decremento possono essere posizionati dopo il nome della variabile (postfissi) oppure prima del nome della variabile (prefissi).
Nel caso seguente l'operatore è postfisso:

```
int num;
int risultato;
num=10;
risultato=(num++) – 3;    //sta a significare che l'incremento di num +1 verrà svolto dopo.
```

L'ultima istruzione equivale a

```
risultato =num-3;  // risultato assume il valore 7
num=num+1; // num assume il valore 11
```

Di seguito viene mostrato l'esempio con l'operatore prefisso.

```
num=10;
risultato=(++num)-3 /*la variabile risultato assume il valore 8, num il valore 11, in quanto l'incremento di num+1 viene svolto prima della sottrazione. */
```

L'ultima istruzione equivale alla seguenti istruzioni;

```
num=num+1;
risultato=num-3;
```

Le strutture di controllo: selezione

La struttura di **selezione** consente di far eseguire particolari istruzioni in alternativa ad altre sulla base del risultato di una condizione. Se la condizione è vera viene eseguito un blocco di istruzione, altrimenti ne viene eseguito un altro. I blocchi sono indicati con le parentesi graffe.

In Java la selezione è realizzata con il seguente costrutto:

```
If  (condizione)
 {
          // istruzioni eseguite se la condizione è vera
  }
else
 {
        //istruzione eseguite se la condizione è falsa
 }
```

Il blocco indicato da else può non essere presente.

Comunque l'uso delle parentesi, anche in presenza di una sola istruzione, serve per ottenere una maggiore leggibilità ed evitare possibili errori di interpretazione.

La condizione che segue l'if può essere una variabile di tipo boolean oppure un'espressione che contiene gli operatori di confronto o gli operatori booleani.

La selezione multipla

La struttura di **selezione multipla** consente di guidare l'esecuzione del programma scegliendo tra diverse alternative, a seconda del valore di una certa espressione.
L'espressione deve restituire un tipo intero oppure carattere.
La struttura generale della selezione multipla in Java è la seguente:

```
swtich  (espressione)
{
  case  valore1:   //istruzione 1
        break;
  case  valore2:   //istruzione 2
        break;
   ……….
   default:  //istruzione n
         break;
}
```

L'espressione dopo la parola chiave **switch** solitamente è una variabile intera, ma può essere anche un carattere o una stringa. All'interno del blocco si elencano i possibili valori che può assumere l'espressione, usando per ogni valore un blocco case. Se non viene trovato alcun valore nell'elenco dei case, si eseguono le istruzioni inserite dopo **default**. La parola chiave **break** serve per indicare la fine del blocco di istruzione e fa terminare la selezione multipla.

Gli operatori di confronto

Gli operatori di confronto sono i seguenti:
L'operatore di uguaglianza è rappresentato da due simboli di uguale (**==**)
La disuguaglianza da due simboli (**!=**) che significano diverso.
Gli operatori di confronto sono:

< , <= , > , >=

Esempio:

```
If  (voto==6)
 {
    System.out.println("sufficiente o più che sufficiente");
  }
Else
 {
    System.out.println("insufficiente");
 }
```

Gli operatori booleani o logici

Gli operatori logici in Java permettono di verificare una condizione.

Operatore	Significato	Uso	Descrizione Restituisce `true` se
&&	AND (calcolo abbreviato) Restituisce `true` se entrambi veri	op1 && op2	op1 e op2 sono entrambi `true`
&	AND Restituisce `true` se entrambi veri	op1 & op2	op1 e op2 sono entrambi `true`
||	OR (calcolo abbreviato) Restituisce `true` se almeno uno è vero	op1 || op2	almeno op1 o op2 `true`
|	OR Restituisce `true` se almeno uno è vero	op1 | op2	almeno op1 o op2 `true`
!	NOT Restituisce `true` se l'operando è `false`, `false` in caso contrario	!op1	op1 è `false`

Esempio:

```
If  ((ora >=12) && (ora<=17))
{
    System.out.println("Buon pomeriggio");
 }
Else
{
    System.out.println("Buona giornata");
 }
```

La selezione multipla

La struttura di **selezione multipla** consente di guidare l'esecuzione del programma scegliendo tra diverse alternative, a seconda del valore di una certa espressione.
L'espressione deve restituire un tipo intero oppure carattere.
La struttura generale della selezione multipla in Java è la seguente:

```
swtich  (espressione)
{
  case  valore1:   //istruzione 1
        break;
  case  valore2:   //istruzione 2
        break;
   ……….
   default:  //istruzione n
         break;
}
```

L'espressione dopo la parola chiave **switch** solitamente è una variabile intera, ma può essere anche un carattere o una stringa. All'interno del blocco si elencano i possibili valori che può assumere l'espressione, usando per ogni valore un blocco case. Se non viene trovato alcun valore nell'elenco dei case, si eseguono le istruzioni inserite dopo **default**. La parola chiave **break** serve per indicare la fine del blocco di istruzione e fa terminare la selezione multipla.

La selezione multipla

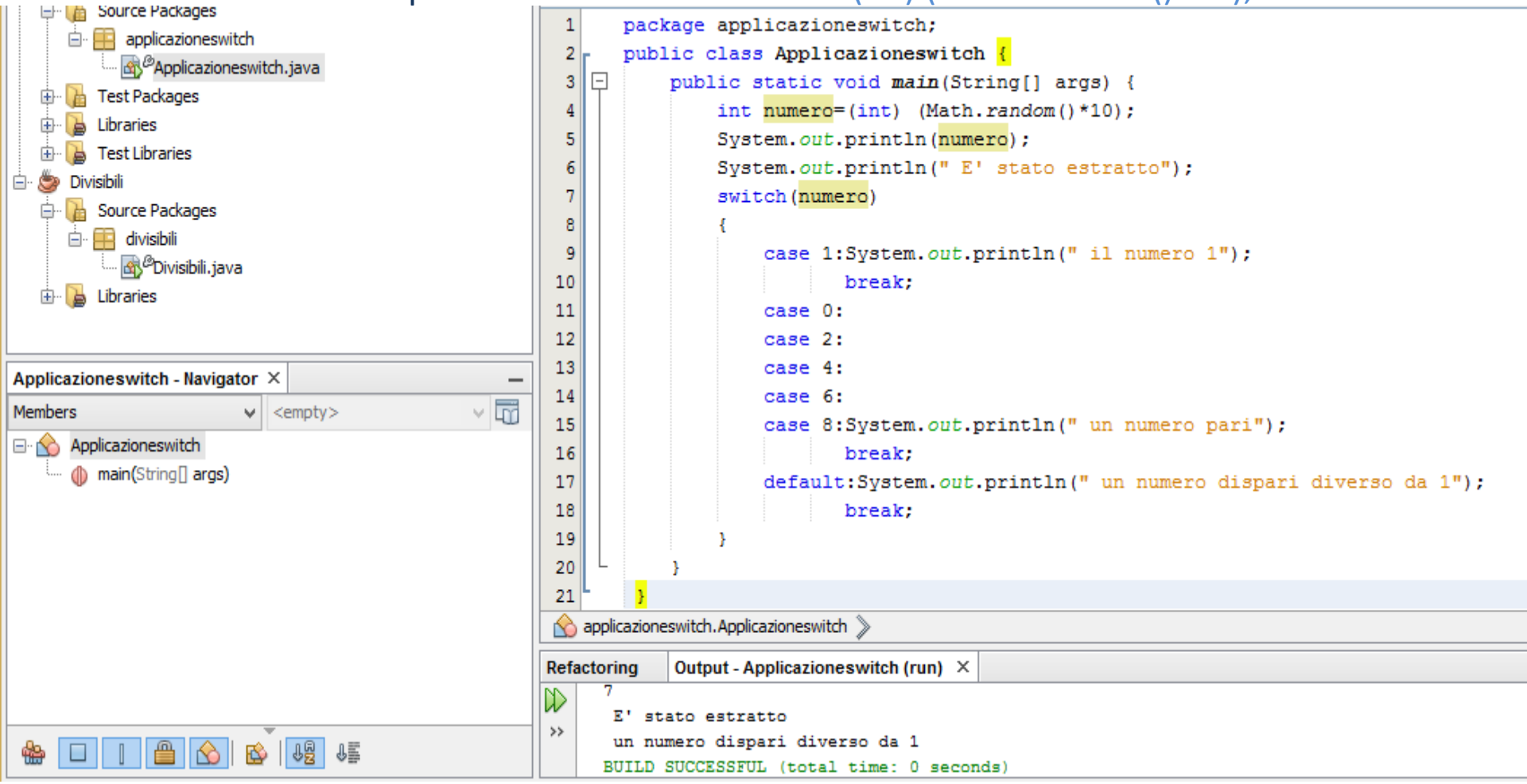
Esempio di selezione multipla.

Nel seguente frammento di programma viene mostrato come funziona il costrutto switch e come sia possibile unire tra loro più casi, associando ad essi un unico blocco di istruzioni

```
package applicazioneswitch;
public class Applicazioneswitch {
   public static void main(String[] args) {
      int numero=(int) (Math.random()*10);
      System.out.println(numero);
      System.out.println(" E' stato estratto");
      switch(numero)
      {
         case 1:System.out.println(" il numero 1");
               break;
         case 0:
         case 2:
         case 4:
         case 6:
         case 8:System.out.println(" un numero pari");
               break;
         default:System.out.println(" un numero dispari diverso da 1");
               break;
      }
   }
}
```

La selezione multipla

Si noti nel programma l'utilizzo del metodo **random** della classe Math, restituisce un numero casuale di tipo double compreso tra 0.0 e 1.0. Moltiplicando il numero restituito per 10 si ottiene un numero compreso tra 0.0 e 10.0. L'operazione di casting converte il numero casuale in un intero troncando la parte decimale: int numero=(int) (Math.random()*10);



Le strutture di controllo: ripetizione in Java

La struttura di ripetizione (o ciclo oppure iterazione) consente di eseguire un blocco di istruzioni più volte. In Java ci sono tre modalità per rappresentare questa struttura di controllo.

While

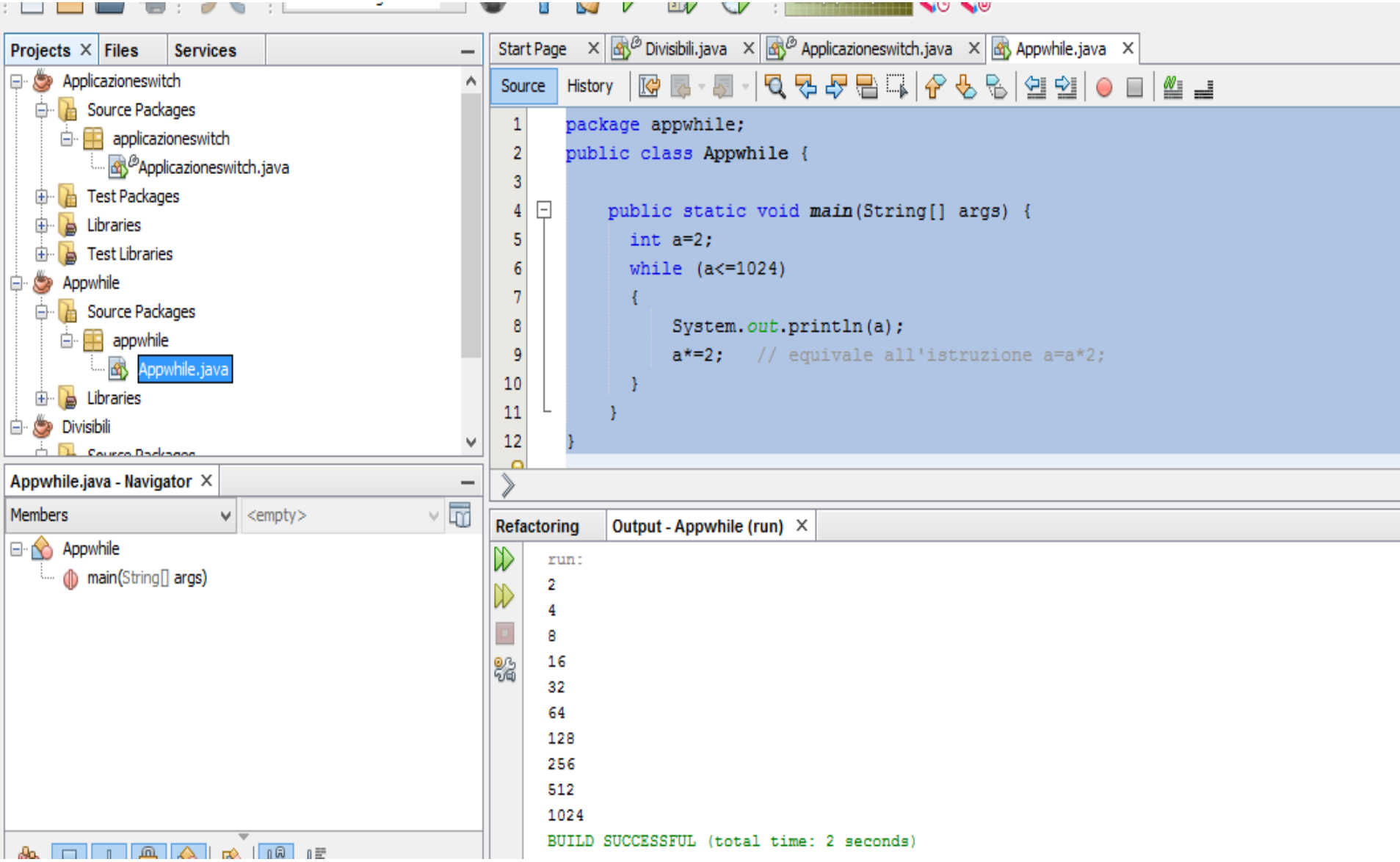
La prima struttura di controllo iterativa assume la seguente forma:

```
While (condizione)
 {
    //istruzione  1
 }
```

Le istruzioni contenute nel blocco vengono eseguite mentre la condizione si mantiene vera. La condizione viene controllata prima di entrare nel ciclo, quindi se è falsa le due istruzioni nel While non vengono eseguite nemmeno una volta. Esempio:

```
package appwhile;
public class Appwhile {
public static void main(String[] args) {
int a=2;
while (a<=1024)
{
   System.out.println(a);
   a*=2;   // equivale all'istruzione a=a*2;
}
}
}
```

WHILE (mentre)



Le strutture di controllo: Do...While

La struttura di ripetizione **do While**

La seconda struttura di controllo iterativa consente di effettuare il test sulla condizione alla fine del ciclo. In questo modo il blocco di istruzione viene eseguito almeno una volta.

La sua struttura è al seguente:

```
do
 {
    //istruzione  1
  }
While (condizione)
```

I cicli infiniti

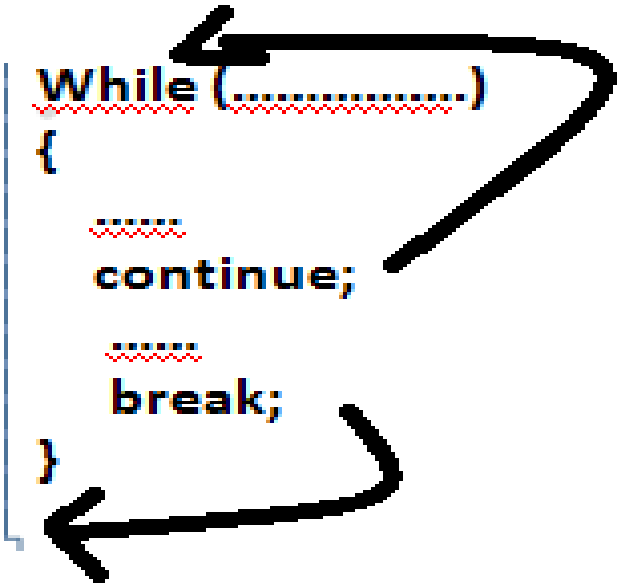
In particolare può capitare di scrivere un programma con strutture iterative che diventano dei cicli infiniti (loop infinito), nei casi in cui la condizione del ciclo si mantenga sempre vera. Anche se si esegue un ciclo infinito è possibile interrompere l'esecuzione del programma premendo la combinazione di tasti CTRL+C.

Il seguente frammento di codice è un esempio di ciclo infinito:

```
do
{
   //istruzione
}
while (true)
```

I cicli interrotti

Le istruzioni presenti nei blocchi delle strutture iterative vengono eseguite in sequenza. E' però possibile modificare l'ordine di esecuzione usando due istruzioni: **break e continue.**



L'istruzione **continue** interrompe l'esecuzione del blocco e la fa riprendere dall'inizio del ciclo dopo aver valutato nuovamente la condizione.
L'istruzione **break** causa l'interruzione anticipata della ripetizione, attivando l'esecuzione della prima istruzione presente dopo la struttura di controllo.

Il ciclo for

La terza struttura di controllo interattiva è il ciclo FOR.
Il for consente di eseguire la ripetizione con un contatore.
Inoltre raggruppa all'inizio del ciclo le fasi di inizializzazione, aggiornamento e controllo della condizione. La struttura generale è la seguente:

```
for (inizializzazione; condizione; aggiornamento)
 {
    //istruzione  1
  }
```

Nella parentesi del for, separati da punto e virgola, vanno inseriti i seguenti elementi:
-La parte di inizializzazione contiene il codice che viene eseguito una sola volta prima di entrare nel ciclo for. Solitamente contiene la dichiarazione di una variabile locale e la rispettiva inizializzazione.
-La parte condizione contiene un'espressione booleana che viene valutata prima di eseguire il blocco di istruzione. Se è falsa, non vengono eseguite le istruzioni presenti nel blocco. Se è vera vengono eseguite le istruzioni nel blocco o ciclo.
-Al termine del ciclo vengono eseguite le istruzioni presenti nella parte aggiornamento e successivamente viene rivalutata la condizione per stabilire se la ripetizione deve continuare.
La seguente struttura for visualizza i primi 10 numeri naturali.

```
for (int i=1;i<=10;i++)
    {
       System.out.println(i);
    }
```

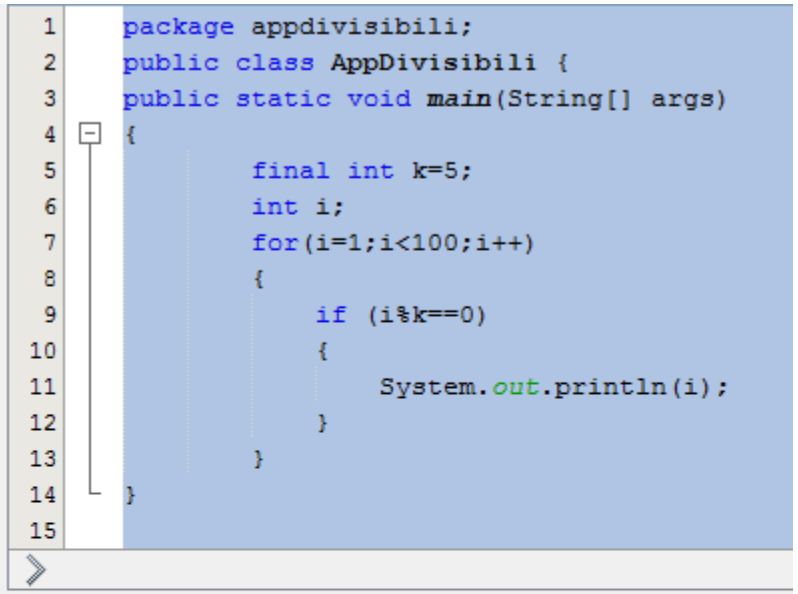
esercizio in Java

Data una costante k, fornire tutti i numeri tra 1 e 100 divisibili per k. Realizzare una classe in Java che visualizzi a schermo tali numeri.

Esempio: se k=5, allora i numeri divisibili tra 1 e 100 saranno 5,10,15,20,25,30,35,……,100

Il sorgente è il seguente:

```
package appdivisibili;
public class AppDivisibili {
public static void main(String[] args)
{
      final int k=5;
      int i;
      for(i=1;i<100;i++)
      {
         if (i%k==0)
         {
            System.out.println(i);
         }
      }
}
}
```



Le operazioni di input

Per le operazioni di **stampa**, output a monitor, abbiamo utilizzato l'oggetto **System.out**, che rappresenta l'oggetto associato allo standard output. Si precisa che esiste anche l'oggetto associato allo standard error, che è **System.err.** Entrambi fanno riferimento, in assenza di una diversa specifica (cioè di default), allo schermo. Sull'oggetto System.out viene invocato il metodo println, con un parametro che rappresenta il valore da visualizzare.

Le operazioni di input

Per le operazioni di input esiste un oggetto analogo **System.in** che gestisce il flusso di dati inseriti da tastiera. L'oggetto **System.in**, in pratica, viene mascherato con altri oggetti più potenti che forniscono maggiori funzionalità.

Si utilizza la classe **BufferedReader** nel seguente modo:

```
InputStreamReader input=new InputStreamReader(System.in);
BufferedReader tastiera= new BufferedReader(input);
```

Con queste dichiarazioni viene definito un oggetto tastiera, di classe BufferedReader, usando l'operatore new che crea un nuovo oggetto.

La classe BufferedReader mette a disposizione il metodo **readLine** che consente di leggere, in questo caso da standard input, una riga per volta.

Una linea viene considerata terminata quando viene premuto il tasto di Invio.

Questo metodo restituisce solo stringhe, quindi se si vogliono acquisire valori numerici si deve effettuare la conversione tra stringhe e numeri.

Le operazioni di input e le eccezioni try…catch

Per effettuare **la lettura di una stringa** occorre dichiarare una variabile stringa e poi usare il metodo readLine con l'oggetto tastiera che è stato definito in precedenza:

```
String nome;
nome=tastiera.readLine();
```

L'operazione deve essere completata considerando le eccezioni che possono essere generate dal metodo readLine. **L'eccezione** segnala una situazione anomala: quando si verifica un'eccezione, bisogna prevedere un blocco di istruzioni per gestire questa situazione. Il costrutto **try…catch…** raggruppa le istruzioni da controllare e dichiara il nome dell'eccezione che si vuole gestire; in particolare la classe **Exception** indica un'eccezione generica. Quindi, nel prossimo programma, proviamo a leggere l'età di una persona, come lettura di una stringa, dopodiché convertiamo la stringa in numero e ne stampiamo il valore. Si ricordi di inserire l'istruzione import java.io.*; dopo package.

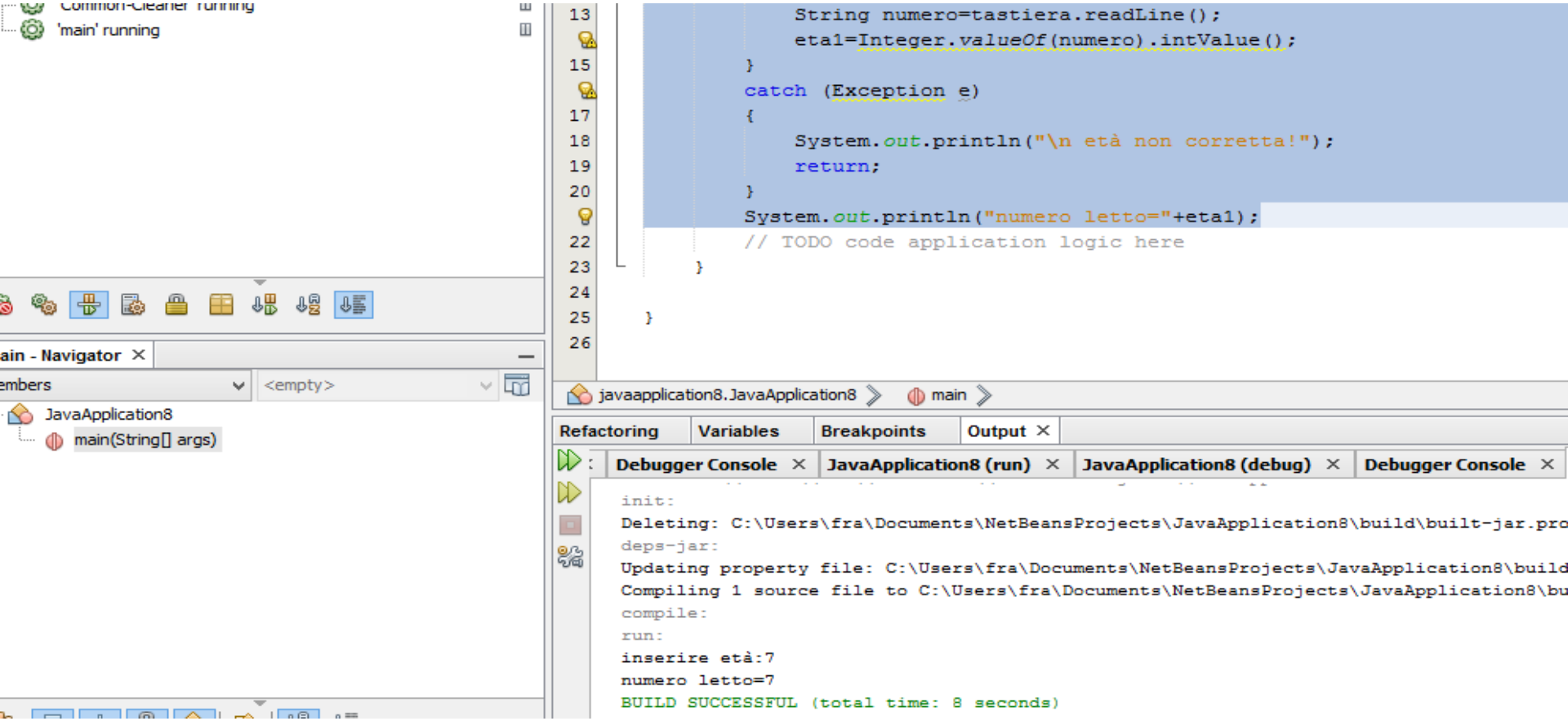
```
InputStreamReader input=new InputStreamReader(System.in);
BufferedReader tastiera= new BufferedReader(input);
int eta1;
System.out.print("inserire età:");
try
{
   String numero=tastiera.readLine();
   eta1=Integer.valueOf(numero).intValue();
}
catch (Exception e)
{
   System.out.println("\n età non corretta!");
   return;
}
System.out.println("numero letto="+eta1);
```

Le operazioni di input

L'istruzione eta1=Integer.valueOf(numero).intValue();

Consente di convertire una stringa in un numero intero, attraverso due metodi della classe **Interger.valueOf** converte il parametro stringa in un oggetto di classe **Integer,** mentre **IntValue** restituisce un valore intero.

In questo caso, l'eccezione che può essere generata si verifica quando i caratteri inseriti non corrispondono a cifre numeriche.



Le operazioni di input

Per leggere un numero in virgola mobile, occorre cambiare il modo di conversione.

Se si vuole ottenere un valore float, si deve scrivere:

```
numFloat=Float.valueOf(leggiNumero).floatValue();
```

In modo del tutto analogo, per ottenere un valore double:

```
numDouble=Double.valueOf(leggiNumero).doubleValue();
```

Esercizio svolto nella pagina successiva.

Leggere da tastiera l'altezza di due persone e calcolare l'altezza media.

Svolgimento esercizio precedente

```
package altezzamedia;
import java.io.*;
public class Altezzamedia {
  public static void main(String[] args)    {
     InputStreamReader input=new InputStreamReader(System.in);
     BufferedReader tastiera= new BufferedReader(input);
     double media,altezza1,altezza2;
     System.out.print("inserire altezza1:");
     try       {
                 String numero=tastiera.readLine();
                  altezza1=Double.valueOf(numero).doubleValue();
                }
     catch (Exception e)       {
                                  System.out.println("\n altezza1 non corretta!");
                                   return;
                                  }
     System.out.print("inserire altezza2:");
     try       {
                 String numero=tastiera.readLine();
                 altezza2=Double.valueOf(numero).doubleValue();
                }
     catch (Exception e)       {
                                   System.out.println("\n altezza2 non corretta!");
                                   return;
                                   }
     media=(altezza1+altezza2)/2;
     System.out.println("altezza media è="+media);
  } }
```

Massimo comune divisore (MCD)

Esercizio: dati due numeri positivi, si deve calcolare il più grande numero positivo che divide entrambi i numeri (MCD).

```
package mcd;
import java.util.Scanner;
public class MCD {
   public static void main(String[] args) {
      int a,b; int scambio; int resto; int mcd;
      Scanner s=new Scanner(System.in); // creo l'oggetto per leggere da tastiera
      a=s.nextInt(); //inserisco primo numero
      b=s.nextInt(); //inserisco secondo numero
      if (a<b)
      { scambio=b;
        b=a;
        a=scambio;
      }
      resto=a%b;
      while (resto!=0)
      {  a=b;
         b=resto;
         resto=a%b;
      }
      mcd=b;
      System.out.println("Massimo comun divisore:"+mcd);
      }
}
```

La gestione degli input tramite la classe Scanner

Per leggere i **numeri e stringhe** si può ricorrere alla classe **Scanner** del package java.util. Vediamo un esempio. Acquisire un intero e una stringa da tastiera e visualizzare a monitor.

```
package leggi;
import java.util.Scanner;
public class Leggi {
  public static void main(String[] args)    {
      Scanner s=new Scanner(System.in); // creo l'oggetto per leggere da tastiera
      int k=s.nextInt(); //inserisco un numero
      System.out.println("Hai digitato il numero:"+k);
      String  string1=s.next();  //inserisco una stringa
      System.out.println("Hai digitato la stringa:"+string1);
  }   }
```

La classe **Scanner** mette a disposizione dei metodi per acquisire da tastiera differenti tipi di elementi:

nextInt() //acquisisce un intero
nextDouble() // acquisisce un double
next() // acquisisce una sola stringa
nextLine() // acquisisce più stringhe separate da spazio poste in un'unica linea.

Librerie e package

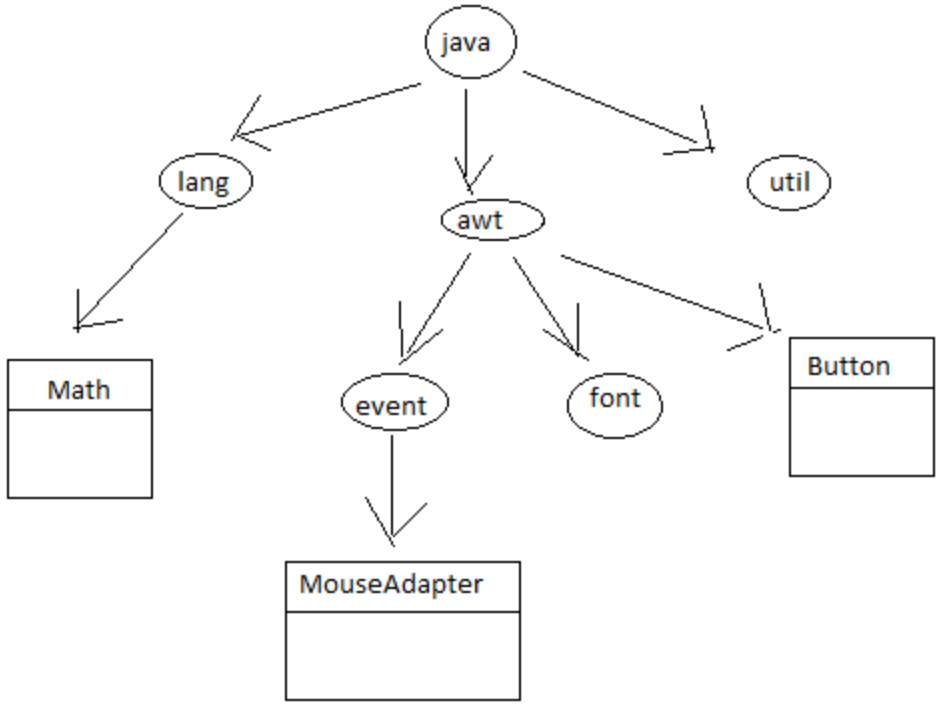
L'ambiente di programmazione Java include un certo numero di package, costituiti ciascuno da un insieme di classi già compilate, che possono essere utilizzate all'interno dei programmi, eliminando la possibilità di conflitti tra nomi di classi in gruppi distinti.
I package di Java sono veramente molti e, per tale motivo, non è possibile descriverli tutti, neppure in modo superficiale: analizzeremo, più o meno in dettaglio, solo quelli che man mano ci serviranno, rimandando per gli altri alla guida in linea del JDK.
Il nucleo del linguaggio Java contiene solo le parole chiave per costruzione delle classi, i commenti, i normali costrutti, la dichiarazione dei tipi primitivi e poco altro.
Alcuni esempi di package e del loro contenuto son presentati nella tabella seguente:

Package	Contenuto
Java.lang	Le classi di base (è incluso per default in tutte le applicazioni)
Java.io	Le classi per la gestione dei file e dei flussi di input e output
Java.awt	Le classi per la gestione dei componenti grafici (colori, font, bottoni, finestre
Java.swing	Le classi per la creazione di interfacce utente grafiche (bottoni, campi di testo, finestre ecc.)
Java.net Java.applet	Il supporto per creare applicazioni che si scambiano dati attraverso la rete. Le classi per la gestione degli applet.

Librerie e package

Le librerie di classi nel JDK sono contenute nel package java: nella gerarchia dei package, si tratta del package capostipite. Il package java contiene infatti a sua volta i package che definiscono il linguaggio, le classi per l'input/output, semplici funzioni per la comunicazione in rete ecc.

I package sono strutturati in forma gerarchica: ogni package può contenere al proprio interno classi oppure altri package: Ad esempio, dal seguente schema notiamo che il package awt è contenuto nel package java e contiene a sua volta i package font e event e la classe Button. Nello schema si utilizzano iniziali **minuscole per riferirsi a package** e **iniziali maiuscole per riferirsi alle classi**.



Librerie e package

Il package awt (Abstract Window Toolkit), mette a disposizione del programmatore una serie di classi e Interfacce per la gestione di interfacce grafiche.
Il package java.awt ha anche dei sottopackage, essi sono:
java.awt.color; java.awt.datatransfer; java.awt.dnd; java.awt.event; **java.awt.font**; java.awt.geom; java.awt.im; java.awt.image; java.awt.image.renderable; java.awt.print

Per importare una singola classe di un determinato package si utilizza la sintassi :
import <percorso dei package>.<nomeclasse>;

Ad esempio: import java.awt.font;
Importa la classe font dal package java.awt

Per importare un intero package di classi si utilizza invece la seguente sintassi:
import <percorso dei package>.*;

Ad esempio:
import java.awt.*;
Importa tutte le classi del Package java.awt (inclusa la classe Font)

Esercizio

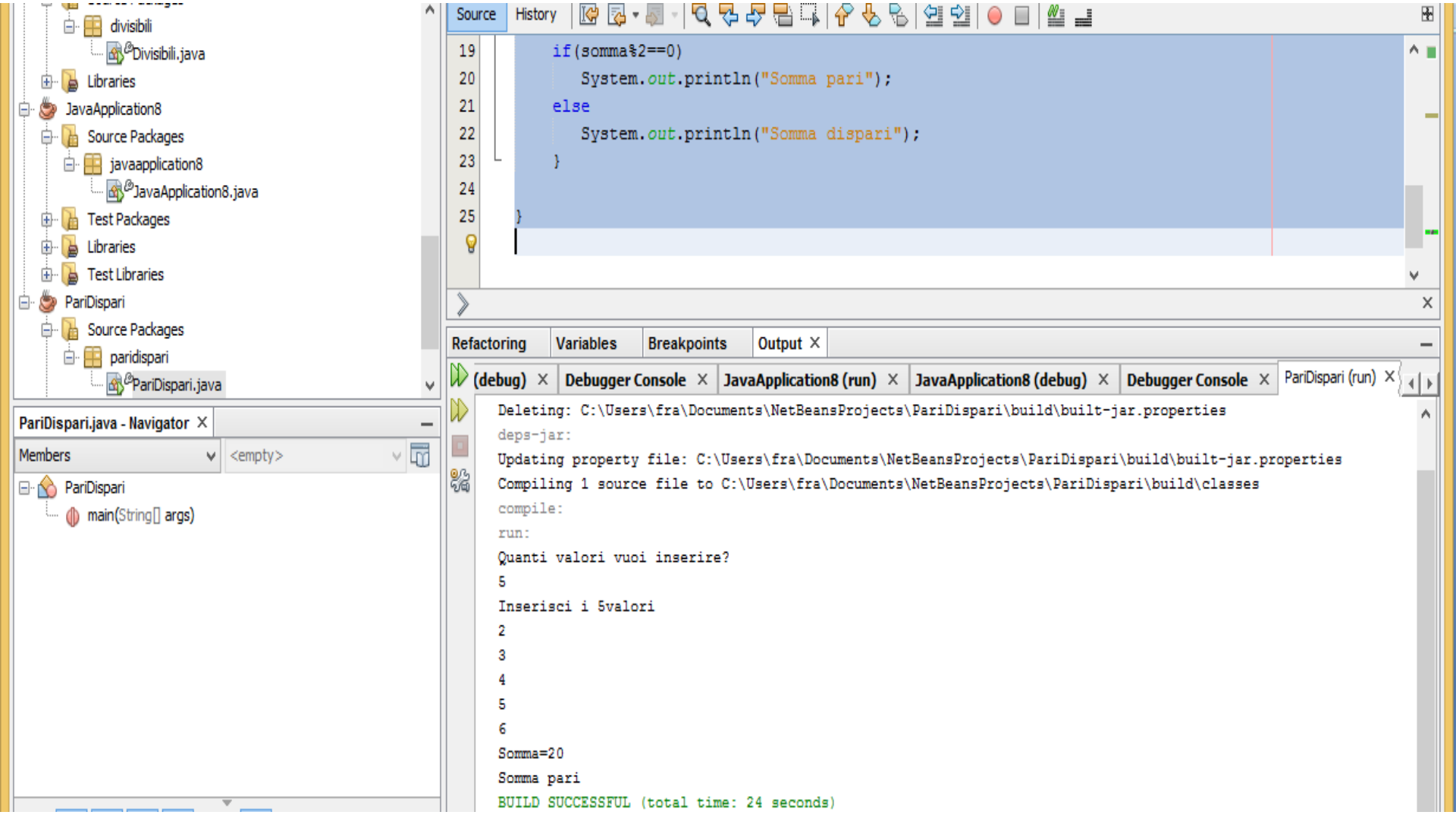
Realizzare un programma che calcoli la somma di n interi acquisiti da tastiera e visualizzi se tale somma è pari oppure dispari.

```
package nparidispari;
import java.util.Scanner;
public class NPariDispari {
   public static void main(String[] args)    {
     int n,v, somma=0;
     Scanner s= new Scanner(System.in);
     // creo lo scanner
     System.out.println("Quanti valori vuoi inserire?");
     n=s.nextInt();
     System.out.println("Inserisci i "+n+"valori");
     for(int i=0;i<n;i++)
     {
        v=s.nextInt();
        somma+=v;
     }
   System.out.println("Somma="+somma);
   if(somma%2==0)
     System.out.println("Somma pari");
   else
     System.out.println("Somma dispari");
   }

}
```

Esercizio

Vediamo il risultato del programma precedente, per correttezza espositiva ne riportiamo di nuovo la traccia.
Realizzare un programma che calcoli la somma di n interi acquisiti da tastiera e visualizzi se tale somma è pari oppure dispari.



Le variabili strutturate: gli array

Quando si devono memorizzare più oggetti che sono dello stesso tipo, si possono utilizzare gli array a una dimensione (**vettori**) o a più dimensioni (**matrici**).

Un **array** è realizzato con un riferimento che punta all'area di memoria dove si trovano i suoi elementi. Per indicare l'array, cioè un gruppo di elementi dello stesso tipo, si usa un unico nome collettivo. Ogni elemento è individuato attraverso l'indice (numero intero). Il primo elemento dell'array ha come indice 0. Il numero di elementi che compongono l'array costituisce la dimensione dell'array. Quindi la posizione dell'ultimo elemento dell'array è individuato dall'indice ottenuto decrementando la dimensione di uno. Per creare e usare un array si devono seguire tre passaggi:

-**Dichiarazione dell'array;**

-**Allocazione;**

-**Eventuale inizializzazione.**

La **dichiarazione** di un array consiste nella specificazione del suo tipo. Il tipo può essere un tipo predefinito di Java (per esempio int) oppure può essere una classe (per esempio String, Automobile, ecc.). Si possono creare array che contengono un insieme di oggetti tutti della stessa classe.

Come per le variabili, anche in questo caso la dichiarazione è fatta indicando un tipo e poi il nome da associare all'array; però subito dopo il nome sono presenti le parentesi quadre per indicare che si tratta di un array.

Le variabili strutturate: gli array

Le parentesi potrebbero essere messe anche dopo il tipo; al momento, per non generare confusione, si userà la dichiarazione con le parentesi dopo il nome.

```
int i[];    // array di interi
String nomi[];  // array di oggetti String
```

L'allocazione

La sola dichiarazione non specifica quale sia la dimensione dell'array. L'array, a questo punto, non è ancora stato creato e assume il valore speciale **null.**

L'allocazione di un array consente di specificare quanto spazio in memoria riservare per l'array, cioè quanti elementi saranno memorizzati al suo interno. L'allocazione viene eseguita utilizzando l'operatore **new.** E' lo stesso operato che sarà utilizzato per allocare gli oggetti. L'operatore new è seguito dal tipo dell'array e dalla dimensione, specificata tra parentesi quadre.

```
i=new int[5];                 // array di 5 interi
nomi=new String[10];          // array di 10 oggetti String
```

Dopo la fase di allocazione, si può dire che l'array è stato creato e può essere utilizzato, assegnando i valori ai suoi elementi.

E' possibile riferirsi ai singoli elementi dell'array utilizzando il nome e un numero che indica la posizione all'interno dell'array. Il numero usato è l'**indice** dell'array.

Le variabili strutturate: gli array

Si parte con l'indice 0 per indicare il primo elemento e si termina con l'indice dato dalla dimensione meno 1 per indicare l'ultimo elemento. Se si cerca di accedere a un elemento che non fa parte dell'array, viene generata l'eccezione **ArrayIndexOutOfBoundsException** e il programma non viene eseguito correttamente.

L'inizializzazione

Per assegnare i valori agli elementi di un array si procede così:

```
i[0]=35;
i[1]=12;
i[2]=3;
nomi[0]="Francesca";
```

Questi assegnamenti se vengono eseguiti subito dopo la dichiarazione e l'allocazione, rappresentano l'inizializzazione dell'array.

Un array mantiene al suo interno l'informazione della sua dimensione. Per richiamare questa informazione si fa seguire al nome dell'array un punto e la parola **length**.

i.length restituisce il numero 5.

nomi.length restituisce il numero 10.

Questo modo di indicare la dimensione dell'array è utile all'interno dei cicli per conoscere la dimensione dell'array.

Le variabili strutturate: gli array

Con il seguente ciclo **for** si assegna a ogni elemento dell'array **i** il valore corrispondente al quadrato del suo indice.

```
for (int indice=0;indice<i.length; indice++)
{
  i[indice]=indice*indice;
}
```

Java consente di unire le due fasi di dichiarazione e allocazione per rendere più veloce la creazione dell'array: dichiarazione e allocazione degli array possono essere fatte contemporaneamente sulla stessa riga.

```
int i[] = new int[5];
String nomi[]= new  String[10];
```

E' possibile anche dichiarare, allocare e allo stesso tempo inizializzare un array, inserendo tra parentesi graffe i valori che deve contenere.

```
int  i[]={45,2,4,10,200};
String nomi[]={"Francesca","Antonio","Marco"};
```

La dimensione dell'array, in questo caso, corrisponde al numero di elementi nelle parentesi graffe.

Esercizio con array

Scrivere un programma java che legga da tastiera 10 nomi di animali da inserire in un array e subito dopo li visualizzi.

```
package arraystring;
import java.util.Scanner;
public class ArrayString {
   public static void main(String[] args) {
      int i=0;
      String animali[]=new String[10];
      Scanner  mio=new Scanner(System.in);
      try
      {
         while(i<animali.length)
         {
            System.out.println("inserire il "+(i+1)+" animale");
            animali[i]=mio.next();
            i++;
         }
         i=0;
         while(i<10)
         {
            System.out.println((i+1)+" elemento: "+animali[i]);
            i++;
         }
      } // (fine try
      catch (Exception e)
      {
         System.out.println("Si è verificata un \' eccezione");
      }// fine catch
      }// fine main    } // fine classe
```

Ordinamento di un array con il metodo Bubble sort

In informatica il **Bubble sort** o ordinamento a bolla è un semplice algoritmo di ordinamento di dati appartenenti ad un vettore. L'algoritmo di ordinamento consiste nel confrontare il primo elemento con il secondo, se il primo elemento è maggiore del secondo, si scambiano di posizione: altrimenti si prosegue confrontando il secondo elemento con il terzo, se il secondo è maggiore del terzo, si scambiano di posizione e così via.

Supponiamo di voler ordinare in modo crescente il seguente array vett[10,6,4,3,2]. Passiamo ora alla descrizione dell'algoritmo.

Dichiariamo il vettore e l'indice i.

int vett[]={10,6,4,3,2}; int i;

(1) Si inizializza l'indice i del vettore a 0 (i=0);

Si confronta l'elemento i-esimo dell'array (zero nel primo caso) con l'elemento successivo, poiché il dieci e maggiore del sei allora si scambiano: per effettuare lo scambio utilizziamo una variabile di appoggio denominata swap.

Quindi, salvo nella variabile swap il valore 6 (ossia vett[i+1]), successivamente salvo in vett[i+1] il valore di vett[i] (cioè 10), dopodiché salvo in vett[i] il valore conservato precedentemente in swap (cioè 6). In tal modo avrò il seguente vett[6,10,4,3,2]. Incremento i di uno, e ripeto le operazioni dal punto **(1)** sopra.

Quando raggiungo la fine dell'array (i=3), l'elemento più grande è stato messo in ultima posizione:

vett [6,4,3,2,10]. Nel primo ciclo avrò eseguito quattro iterazioni: è importante fermarsi quando i=3, in quanto verrà confrontato l'ultimo elemento con quello successivo (i=4).

A questo punto riparto dal punto (1) e ripeto la stessa procedura fermandomi al penultimo elemento dell'array (in tal modo avrò salvato sull'ultima posizione del vettore il valore più grande e sulla penultima il secondo valore più grande).

vett [4,3,2,6,10].

Ripeto le iterazioni spostando ogni volta l'estremo verso sinistra,fino ad arrivare a due elementi. Alla fine otterrò il vettore ordinato vett[2,3,4,6,10].

```
package bubblesort;          //Algoritmo ordinamento di un array con il metodo Bubble sort
public class BubbleSort {
   public static void main(String[] args)     {
      int  vett[]={10,6,4,3,2}; int n=5; int swap; int i=0; int j=0; int m;
      m=n;
      while(n>1)
      {
         for (j = 0; j < n-1; j++)
            {
               if (vett[j] > vett[j+1])
                  { //Scambio tra i due elementi
                     swap= vett[j+1];
                     vett[j+1] = vett[j];
                     vett[j] = swap;
                  }
            }
         n=n-1;
      }  // stampa vettore ordinato
      for(j=0;j<m;j++)
         {
            System.out.println(vett[j]);
         }
   }// termine del main
 }// termine classe
```

Esercizio: Verificare se un vettore di caratteri è palindromo (cioè se la parola risulta identica leggendola da destra a sinistra e da sinistra a destra)

```
package palindroma;
import java.util.Scanner;
public class Palindroma {
   public static void main(String[] args)   {
      int i=0; int lunghezza=0; int pal=0; String parola;
      Scanner s=new Scanner(System.in); // creo l'oggetto per leggere da tastiera
      parola=s.next();  //inserisco una stringa
      System.out.println("la lunghezza della stringa inserita è:"+parola.length());
      lunghezza=parola.length();  //calcolo la lunghezza della parola inserita
      i=0;
      do
      {
         if (parola.charAt(i)!=parola.charAt(lunghezza-1))    //il metodo charAt ritorna il carattere all'indice specificato
               {
                  pal=1;
               }
         i++;
         lunghezza--;
      } while (i<lunghezza);
      if (pal==1)
            {
               System.out.println("la stringa "+parola+" non è palindroma");
            }
      else
      {
         System.out.println("la stringa "+parola+" è palindroma");
      }
   } // fine main
}  //fine classe
```

Gli array multidimensionali

Gli array considerati finora corrispondono in matematica ai vettori.

Esistono altri tipi di array chiamati multidimensionali, in particolare l'array bidimensionale,che corrisponde in termini matematici a una matrice con righe e colonne. Ogni elemento della matrice è individuato da una coppia di numeri di cui il primo indica la riga e il secondo la colonna.

Lo schema seguente presenta gli indici riga e colonna di una matrice di dimensione 3x4

0,0 0,1 0,2 0,3
1,0 1,1 1,2 1,3
2,0 2,1 2,2 2,3

In java la dichiarazione e l'allocazione di una matrice avviene allo stesso modo della dichiarazione di un array: occorre specificare la nuova dimensione usando un secondo paio di parentesi quadre.

Per assegnare i valori alla matrice bisogna riferirsi a uno specifico elemento della matrice indicando la posizione con una coppia di indici (la riga e la colonna).

```
matrice[0][0]=10;
```
Assegna il valore 10 al primo elemento della matrice

Per dichiarare e creare una matrice 5 righe per 2 colonne che contiene i soli dati di temperatura scriveremo:

```
double temperature[][]= new double[5][4];
```

Per assegnare ad una variabile x il valore della prima cella della matrice scriveremo:

```
double x=temperature[0][0];
```

Popolare una matrice di temperatura di 5 righe e 2 colonne e calcolare la temperatura media della settimana

```
double temperature[][] = new double [5][2];
Scanner  Input=new Scanner(System.in);
int i=0,j;
double somma=0;
double media=0;
try
{  while(i<temperature.length)
   {   j=0;
       while (j<temperature[i].length)
       {
        System.out.print("Temperatura  alla  "+i+"° riga e "+j+"colonna");
        temperature[i][j]=Input.nextDouble();
        j++;
       }// fine  while interno
       i++;
   }// fine while esterno
   i=0;
   while(i<temperature.length)
   {   j=0;
       while(j<temperature[i].length)
       {
somma=somma+temperature[i][j];
        j++;
       }//fine while interno
       i++;
      }//fine while esterno

media=somma/(temperature.length*temperature[0].length);
      System.out.print("la      media è"+media);
   }// fine try
   catch (Exception e)
   {
    System.out.println(" Si  è  verifica un\'eccezione ");
   }//fine catch
 }//fine main  }
```

La matrice trasposta

Esercizio
Dopo aver costruito una matrice 3x3, inizializzata con valori casuali, calcolare la matrice trasposta.

La matrice trasposta è una matrice avente le stesse dimensioni, ma righe e colonne scambiate tra di loro.

Esempio:

Matrice originale
9 1 6
5 5 4
1 9 8

Matrice trasposta
9 5 1
1 5 9
6 4 8

Codice sorgente - matrice trasposta

```
package matricetrasposta;
public class MatriceTrasposta {
   public static void main(String[] args) {
     int matr[][]=new int[3][3]; //dichiarazione ed allocazione di due matrici 3x3
     int trasp[][]=new int[3][3];
     System.out.println("Matrice originale *******");
     for(int riga=0;riga<3;riga++) //inizializzazione casuale della prima matrice
     {
       for(int colonna=0;colonna<3;colonna++)
       {
         matr[riga][colonna]=(int) (Math.random()*10);
         System.out.print(matr[riga][colonna]+" ");
       }
       System.out.println();
     }
     System.out.println("Matrice trasposta *******"); //calcolo della trasposta
     for(int riga=0;riga<3;riga++)
     {
       for(int colonna=0;colonna<3;colonna++)
       {
         trasp[riga][colonna]=matr[colonna][riga];
         System.out.print(trasp[riga][colonna]+" ");
       }
       System.out.println();
     }
   } }
```

Esercizi da svolgere

- Scrivi un programma che stampi i primi 100 numeri pari usando la struttura while.
- Scrivi un programma che stampi i primi 100 numeri dispari usando la struttura do..while.
- Scrivi un programma che stampi i primi 100 numeri pari usando la struttura for.
- Dopo aver caricato in memoria un array di numeri interi con 10 componenti, conta e stampa le componenti che hanno valore superiore a 18.
- Esegui una ricerca in un array: dopo aver generato un array di 30 numeri casuali tra 0 e 99, determina se è presente il numero 50.
- Scrivere un programma che produca a video la tavola pitagorica.
- Leggere una matrice quadrata, di numeri interi, sommare tutti gli elementi della diagonale principale. Stampare la matrice e la somma ottenuta.
- Leggere una matrice di interi, sommare tutti gli elementi della diagonale secondaria. Stampare la matrice e la somma ottenuta.
- Leggere un vettore di N numeri interi, contare quanti numeri pari ci sono nel vettore e stampare il risultato.
- Leggere un vettore di N numeri reali, calcolare e stampare la media ottenuta.

Esercizi da svolgere

- Scrivere un programma che calcoli il prodotto tra due matrici A e B di dimensioni rispettivamente 3x2 e 2x3. Stampare le matrici A e B e la matrice prodotto.

  esempio:

Matrice A			Matrice B				Matrice Prodotto				Matrice Prodotto		
1	0		2	1	4		((1*2)+(0*1))	((1*1)+(0*0))	((1*4)+(0*3))		2	1	4
2	0	x	1	0	3	=	((2*2)+(0*1))	((2*1)+(0*0))	((2*4)+(0*3))	->	4	2	8
0	2						((0*2)+(2*1))	((0*1)+(2*0))	((0*4)+(2*3))		2	0	6

- Dato un vettore di N numeri interi, cancellare l'elemento di cui si conosce la posizione e ricompattare il vettore. Esempio: vettore originale={1,22,31,4,5}; l'elemento da cancellare è nella posizione 3; nuovo vettore ottenuto dopo la cancellazione: {1,22,4,5}.
- Fusione. Realizzare un algoritmo che, dati in input due vettori disordinati di N e M componenti rispettivamente (con N e M<100), li ordini e li fonda in un unico vettore che dovrà risultare anch'esso ordinato in modo crescente.
- Dato un vettore di interi, cancellare i dati contigui ripetuti, compattare a sinistra. Visualizzare il numero di elementi cancellati e la lunghezza del vettore risultante.
- Dato un numero intero minore di 256, convertire il numero decimale in numero binario. Salvare il numero binario ottenuto dalla conversione in un vettore di 8 elementi interi. Stampare il numero decimale ed il vettore di numeri binari.
- Scambia gli elementi di un vettore in modo simmetrico (senza utilizzare un altro vettore).
- Caricare un vettore di N elementi, con N<100, e successivamente trova:

  -la componente più grande, e quella più piccola e le rispettive posizioni occupate nel vettore;

  -la media delle componenti;

  -lo scarto di ogni componente dalla media (salvare gli scarti in un secondo vettore e stamparlo).
- Data una matrice NxM, determinare se gli elementi in posizione pari sono compresi tra -3 e 25 e gli elementi in posizione dispari sono multipli di 3.
- Data una matrice 4x4, riempi con il valore 1 le colonne di posizione dispari e con il valore 0 gli elementi di posizione pari.

Esercizi da svolgere

- Riempire con il valore 0 la diagonale secondaria di una matrice 5x5, riempire tutti gli altri elementi con valore 1.
- Scrivere un programma che, data una stringa, restituisca il numero di vocali contenute in essa.
- Scrivere un programma che, date due stringhe, controlli se la seconda è contenuta nella prima.
- Scriver un programma che, date due matrici A e B di dimensioni N x M, salvi in due vettori gli elementi delle diagonali principali delle due matrici. Stampare Le matrici e i due vettori.
- Dato un vettore di interi, ordinare il vettore in modo decrescente. Stampare il vettore originale e quello ordinato.
- Moltiplicare un vettore di N numeri interi per una Matrice di N righe ed M colonne. Stampare le strutture originali ed il prodotto ottenuto.
- Leggere una matrice quadrata , quindi scambiare tutte le colonne pari con quelle dispari. Stampare la matrice originale e quella modificata.
- Leggere una matrice quadrata, quindi scambiare le due diagonali. Stampare la matrice originale e quella modificata.
- Caricare una matrice di caratteri, copiare in un vettore tutte le vocali trovate nella matrice precedentemente caricata. Stampare il vettore e la matrice.