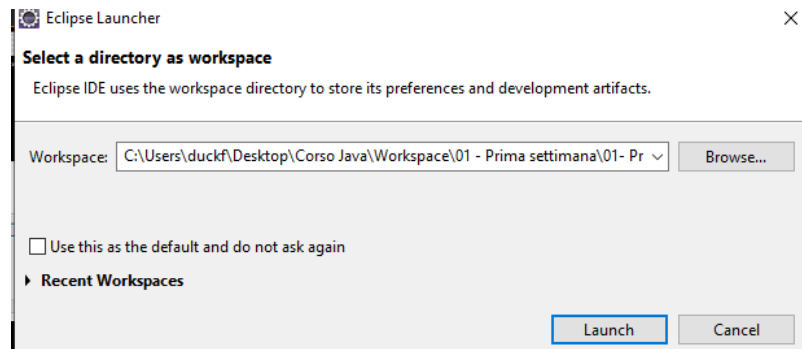


Sommario

1.0 – INTRODUZIONE A ECLIPSE	1
1.1 CREARE NUOVO PROGETTO	2
2. JAVA	6
2.1 Sintassi e cose utili	6
2.1.2 – Operatori matematici	6
2.1.3 – Operatori logici	6
2.1.4 – Convertire il case	6
2.1.5 – Modulo	7
2.2 I tipi	7
2.2.1 – Lo scope di una variabile	8
2.3 Le variabili	9
2.4 Principi della programmazione	9
2.5 Convertire string in numeri	9
2.6 Lo scanner e gli input dell'utente	9
2.7 Secondo principio della programmazione – la selezione	12
2.7.1 – If piatto	12
2.7.2 – If else	13
2.7.3 – If annidati	13
2.7.4 – Operatore ternario	13
2.7.5 – Switch	14
2.8 Terzo principio della programmazione – iterazione	15
2.8.1 – Ciclo While	15
2.8.2 – Ciclo do while	16
2.8.3 – Ciclo for	16
2.8.4 – Differenze tra diversi cicli	17
2.9 Leggere file	17
2.10 – I vettori	19

1.0 – INTRODUZIONE A ECLIPSE

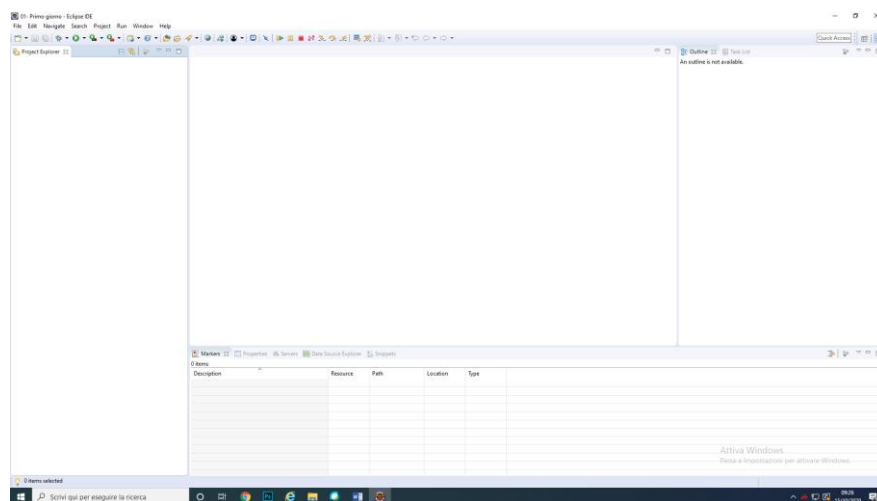
Eclipse è l'ide (integrated development environment) che useremo nel corso. Un IDE è un ambiente di sviluppo.



All'avvio, Eclipse ha bisogno di selezionare un **workspace**, ovvero lo spazio dove vengono salvati tutti i dati di ciò che poi viene eseguito.

Importante è lasciare bianca la spunta di default, in modo che ogni volta si possa selezionare la cartella per il workspace. Una volta selezionata la cartella lancio.

Eclipse crea in automatico la cartella metadata, in cui sono salvati i dati che Eclipse usa per lavorare.



Nella finestra a sinistra, il **project explorer**, si vedono i progetti che ho.

Outline e task list al momento non ci interessano, quindi possiamo toglierli per lasciare spazio al codice. Allo stesso modo la finestra sotto.

Le schede fondamentali sono:

- Quella del codice;
- La console, in cui vedo l'esecuzione del mio codice, il risultato.

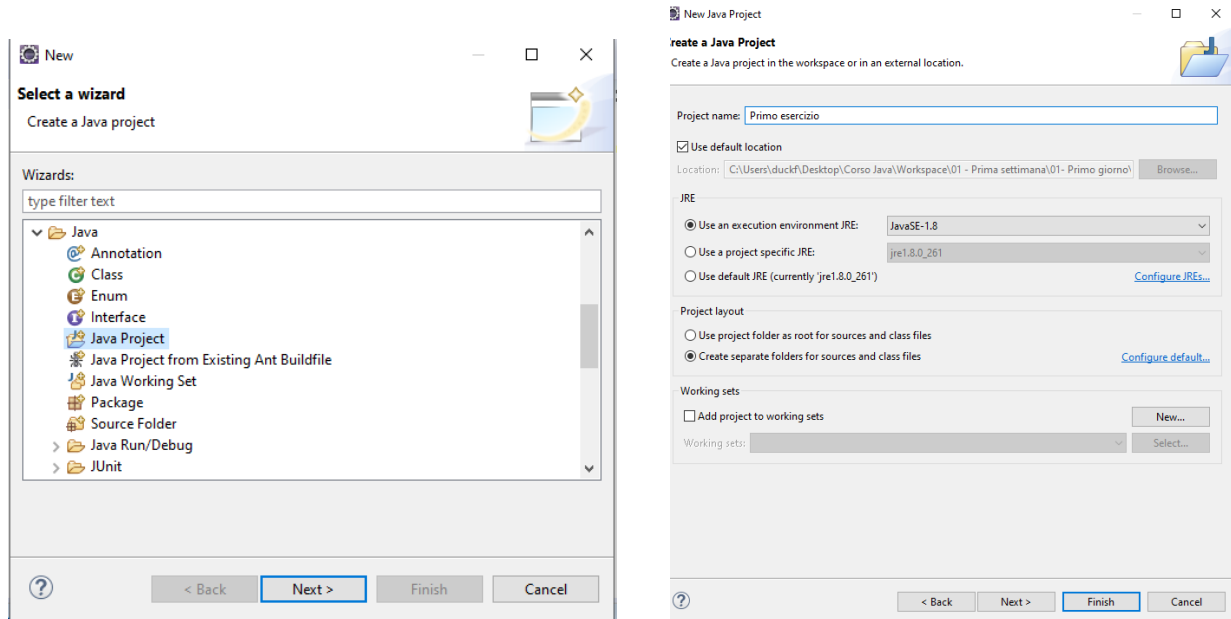
Una **virtual machine** è una macchina virtuale che mi permette di eseguire il codice. Il risultato dell'esecuzione si vede nella console.

1.1 CREARE NUOVO PROGETTO

Per creare un nuovo progetto devo fare:

file -> new -> other

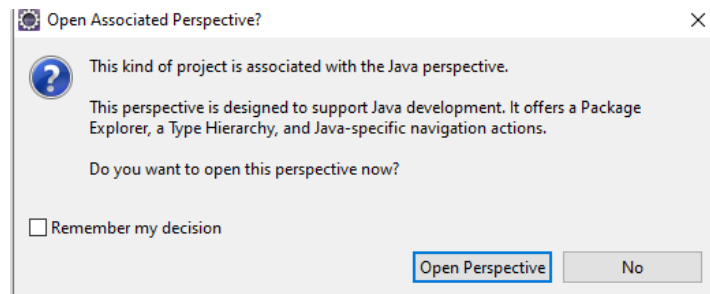
A questo punto ci sono vari tipi di progetti che vengono dati in automatico, noi dobbiamo fare “java project”, siccome non c’è subito in vista, dobbiamo cercare in “altro”.



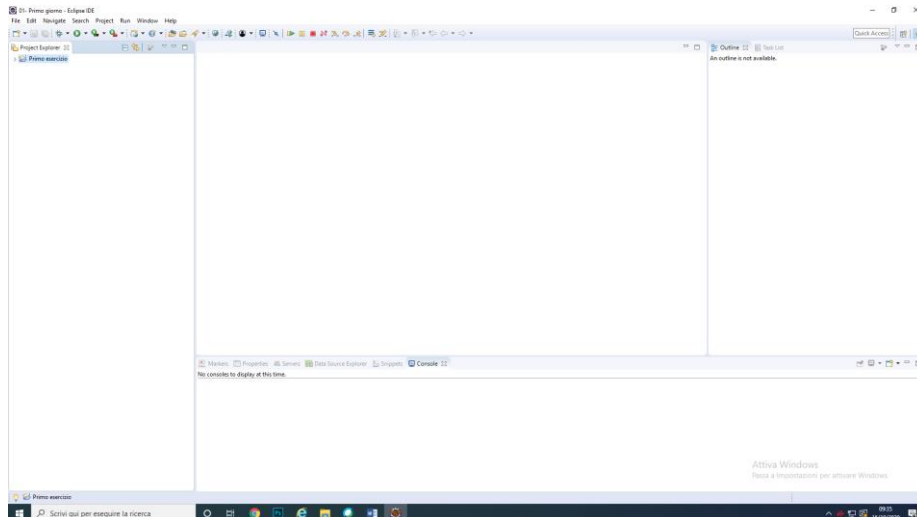
Per ora devo:

- 1- Impostare il nome del progetto;
- 2- Usare “default location”= salva il progetto utilizzando la cartella di riferimento del workspace, data quando ho aperto Eclipse.

A questo punto lancio e viene fuori questo messaggio:








Dice che chi fa progetti in Java, normalmente usa un workspace diverso. Chiede se voglio che apra la scrivania come dovrebbe essere di default. Diciamo di no, così viene mantenuta la scrivania come l’abbiamo impostata prima.



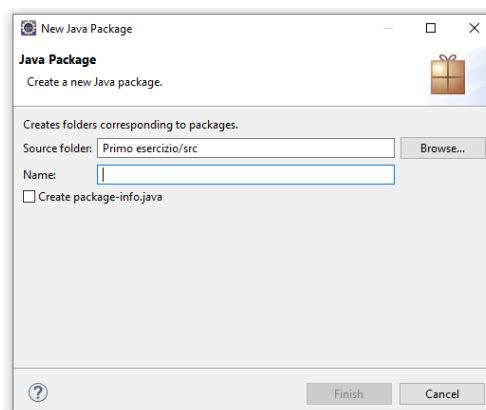
Siamo a questo punto: abbiamo creato il primo grande contenitore, la cartella in cui saranno salvati i dati.

La cartella è strutturata in questo modo:

 .settings	15/10/2020 09:33	Cartella di file	
 bin	15/10/2020 09:33	Cartella di file	
 src	15/10/2020 09:33	Cartella di file	
 .classpath	15/10/2020 09:33	File CLASSPATH	1 KB
 .project	15/10/2020 09:33	File PROJECT	1 KB

Dentro a “src” vanno messi tutti i file che creo.

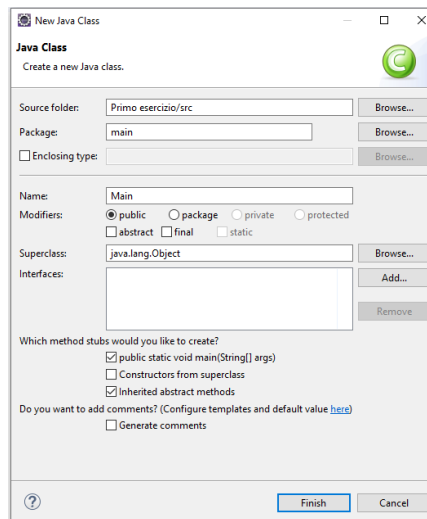
La cartella “Jre system library” contiene le librerie installate, la virtual machine ecc.



Creo un nuovo **package**, ovvero un contenitore. Devo dire dove inserisco il package e poi inserire il nome.

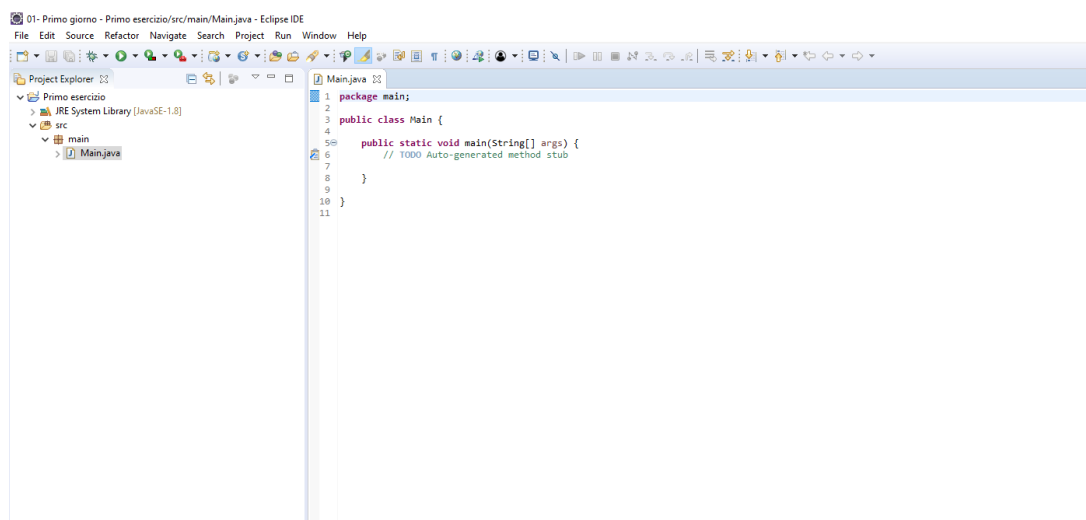
Creo un package di nome “main” dentro alla cartella “src”. Per convenzione il nome dei package è sempre scritto con la lettera iniziale minuscola.

Dentro al package “main” creo una nuova classe:



La classe invece va nominata sempre con la lettera iniziale maiuscola, per convenzione. Per ora non tocco altro. Devo solo spuntare il “Public static void main”.

A questo punto ho creato la classe:



Il package diventa marrone, perché ora contiene una classe.

Una **classe** è un contenitore, in questo caso si tratta di una **classe di avvio**.

“Public static void main args”= è un codice che permette di eseguire questo file. Vuol dire che tutto ciò che sta dentro alle graffe può essere eseguito. Per questo si chiama “classe d’avvio”.

```
1 package main;
2
3 public class Main
4 {
5     public static void main(String[] args)
6     {
7         System.out.println("Ciao mondo!");
8     }
9 }
10
```

```
<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_261\bin\javaw.exe (15 ott 2020, 09:52:09)
Ciao mondo!
|
```

In console mi compare il risultato.

2. JAVA

2.1 Sintassi e cose utili

- Con `//` apro un commento in una sola riga.
- Con `/* */` apro e chiudo un commento lungo più righe.
- `+=` serve a sommare il valore precedente di una variabile a nuove cose.
- `\n` serve per andare a capo.

2.1.2 – Operatori matematici

- `==` indica uguaglianza;
- `>=` indica maggiore o uguale;
- `<=` indica minore o uguale;
- `>` indica maggiore;
- `<` indica minore;
- `!=` indica diverso;

Attenzione, `==` vale per i numeri e i boolean ma non per le String. Se bisogna fare una comparazione tra String si possono usare:

- `.equals` per comparazione esatta;
- `.equalsIgnoreCase` per comparazione che non tiene conto di maiuscole o minuscole;

2.1.3 – Operatori logici

- `||` sta per “or”, basta che una delle due condizioni sia vera per rendere vera la condizione;
- `&&` sta per “and”, entrambe le condizioni devono essere vere per rendere vera la condizione;

2.1.4 – Convertire il case

- `.toLowerCase()` = trasforma tutte le lettere di una parola in minuscolo;
- `.toUpperCase()` = trasforma tutte le lettere di una parola in maiuscolo;

```
if(destinazione.equalsIgnoreCase("oslo"))
    costoBiglietto = 180;
|
```

In questo modo Java confronta il contenuto della stringa senza badare al modo in cui è stata scritta.

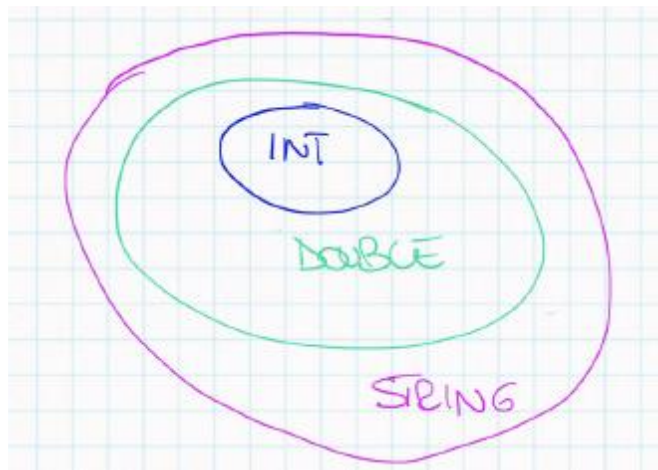
2.1.5 – Modulo

```
String numeriPari = "";
String numeriDispari = "";
for(int i = 0; i < numeri.length; i++) {
    if((i % 2) == 0)
        numeriPari += numeri[i];
    else
        numeriDispari += numeri[i];
}
```

Qui abbiamo un esempio di modulo ($i \% 2$). Il %, a differenza della divisione /, restituisce solo il resto di una divisione.

In questo modo posso determinare se un numero è pari (resto = 0) o dispari (resto != 0).

2.2 I tipi



In Java abbiamo vari tipi di dati:

- **String** = dati di tipo testuale, è l'insieme più grande di dati, che contiene tutti gli altri;
- **int** = numeri interi;
- **double** = numeri decimali
- **char** = singola lettera.
- **boolean** = è un tipo di variabile che può restituire solo due valori (true o false). Si usa di solito quando ci aspettiamo che l'utente risponda sì/no, vero/falso, positivo/negativo.

```
// Dichiarazione  
boolean verifica;
```

```
// Inizializzazione  
verifica = false;
```

Solitamente un boolean viene sempre inizializzato a “false”, in questo modo, se poi viene inserito in un if, sarà la condizione dell’if a determinarne l’eventuale cambiamento di valore.

```
// Dichiarazione  
int numero;  
boolean verifica;  
  
// Inizializzazione  
numero = 10;  
verifica = false;  
  
// Calcolo  
if(numero > 10)  
    verifica = true;  
  
// Output  
System.out.println(verifica);
```

```
String esempioStringa = "Ciao";  
int esempioInt = 3;  
double esempioDouble = 2.3;  
char esempioChar = 'i';
```

Una variabile può contenere un solo tipo alla volta. Può essere che due tipi diversi vengano concatenati ma il risultato è una String.

```
String esempioConcat = "la mia età è" + 32 + "anni";
```

2.2.1 – Lo scope di una variabile

```
case "marinara" :  
    prezzo = 4;  
    int numero = 10;  
    System.out.println(numero);  
break;  
case "esplosiva" :  
    System.out.println(pizza);  
    numero = 12;  
    System.out.println(numero);  
    prezzo = 10;
```


È lo scopo della variabile. La variabile in questo caso si chiama allo stesso modo ma in realtà è dentro al singolo case. Per averla in ogni case dovrei inizializzarla fuori dallo switch. Lo scope quindi è la visibilità di una variabile, per capire quando la posso utilizzare. Lo scope è il punto in cui quella variabile c'è, esiste e la posso usare.

2.3 Le variabili

```
String esempioStringa = "Ciao";  
int esempioInt = 3;  
double esempioDouble = 2.3;  
char esempioChar = 'i';
```

Questi sono esempi di variabili.

Importante è distinguere:

- **Dichiarazione** della variabile = dichiaro una variabile di un certo tipo e con un certo nome, a livello di macchina sto occupando uno spazio nella memoria del computer;
- **Inizializzazione** della variabile = è il momento in cui la variabile viene effettivamente creata, cioè le viene assegnato un valore iniziale. Non ho più solo la scatola ma inserisco qualcosa nella scatola.

```
String esempioStringa;  
  
esempioStringa = "Ciao!";
```

2.4 Principi della programmazione

- **Principio di sequenzialità** = Java legge il programma in sequenza, dalla prima all'ultima riga.
- **Metodo di lavoro DICO** = dichiarazione, inizializzazione, calcolo, output.
- **Selezione** = in un sistema comune dobbiamo avere la possibilità di gestire il dato in entrata e valutarlo prima di autenticarlo.
- **Iterazione** = ripetere un'istruzione o un blocco di codice finché una condizione è vera.
- **DRY** = don't repeat yourself.

2.5 Convertire string in numeri

- **Integer.parseInt()** = serve a convertire una stringa numerica in integer;
- **Double.parseDouble()** = serve a convertire una stringa numerica in Double.

2.6 Lo scanner e gli input dell'utente

Lo **Scanner** è un elemento di Java che permette al programma di stare in ascolto. Può ascoltare la tastiera, un DB, un file, ecc.

Lo Scanner quindi rileva gli input provenienti dall'esterno. Tutti gli input che provengono dall'esterno sono sotto forma di String.

Lo Scanner non è una cosa che Java conosce di default, ogni volta devo farglielo imparare. Bisogna importare una libreria dello Scanner.

```
1 package main;
2
3 import java.util.Scanner;
4
5 public class GeometriaInput
6 {
7     public static void main(String[] args)
8     {
9         // Chiedere all'utente come si chiama
10        // Chiedere all'utente la misura della base e
11        // dell'altezza del rettangolo.
12        // Calcolare e stampare la misura dell'area
13
14        // Dichiarazione
15
16        Scanner tastiera;
17
18    }
19 }
20
```

In questo caso vedo che ho importato il pacchetto corretto.

Alcune librerie vengono importate di default per ogni progetto Java. Altre vanno importate all'occorrenza, lo Scanner è uno degli elementi da importare.

Per ora però l'ho solo dichiarato, bisogna iniziarlo:

```
// Inizializzazione
tastiera = new Scanner(System.in);
```

Questa è la riga per inizializzare lo Scanner, così Java sa che deve mettersi in ascolto dell'input proveniente dalla tastiera dell'utente. Il fatto che la variabile si chiami "tastiera" è parlante ma non è necessario. Una volta eseguita la riga, Java rimane in attesa dell'input dell'utente.

Questa riga serve per gli input da tastiera, se voglio leggere input da DB o altre fonti sarà diverso.

```
// Dichiarazione
Scanner tastiera;
String nomeUtente;
double baseRettangolo, area;
int altezzaRettangolo;
String risposta;

// Inizializzazione
tastiera = new Scanner(System.in);
risposta = "";

System.out.println("Come ti chiami?");
nomeUtente = tastiera.nextLine();
```

Con "tastiera.nextLine()" dico di leggere l'input che l'utente scrive dopo la linea precedente.

```
System.out.println("Ciao " + nomeUtente + "inserisci il valore della base in cm!");  
base Rettangolo = Double.parseDouble(tastiera.nextLine());
```

In questo caso faccio inserire un valore numerico, però tutti gli input sono String, perciò avrò bisogno di convertirlo.

```
int nuovoNumero =tastiera.nextInt();  
double nuovoNumero2 =tastiera.nextDouble();  
|
```

Potrei fare anche in questo modo. Queste due righe fanno la stessa cosa del parseInt e parseDouble, però lo fanno peggio, nel senso che con parseInt e parseDouble viene presa la stringa e viene convertita.

Con nextInt e nextDouble invece viene convertita direttamente senza preoccuparsi di cosa ci sia dentro, il che può portare a errori. Quindi meglio usare parseInt e parseDouble.

```
tastiera.close();|
```

Fondamentale alla fine del programma è inserire questa riga, per indicare a Java che deve smettere di attendere input.

Il canale di immissione dati tra esterno e interno va chiuso perché si consuma una risorsa inutilmente. Se si termina il programma ma la tastiera non è chiusa, potrebbe dare problemi in futuro.

Questo è un esempio di programma completo:

```
public static void main(String[] args)  
{  
    // Chiedere all'utente come si chiama  
    // Chiedere all'utente la misura della base e  
    // dell'altezza del rettangolo.  
    // Calcolare e stampare la misura dell'area  
  
    // Dichiarazione  
  
    Scanner tastiera;  
    String nomeUtente;  
    double base Rettangolo, area;  
    int altezza Rettangolo;  
    String risposta;  
  
    // Inizializzazione  
  
    tastiera = new Scanner(System.in);  
    risposta = "";  
  
    System.out.println("Come ti chiami?");  
  
    nomeUtente =tastiera.nextLine();  
  
    System.out.println("Ciao " + nomeUtente + " inserisci il valore della base in cm!");  
    base Rettangolo = Double.parseDouble(tastiera.nextLine());  
  
    System.out.println("La base del rettangolo è " + base Rettangolo + ". Inserisci il valore dell'altezza in cm!");  
    altezza Rettangolo = Integer.parseInt(tastiera.nextLine());  
  
    System.out.println("L'altezza del rettangolo è " + altezza Rettangolo);  
  
    // Calcolo  
  
    area = base Rettangolo * altezza Rettangolo;  
  
    risposta = "L'area del rettangolo di base " + base Rettangolo + " e altezza " + altezza Rettangolo + " è: " + area;  
  
    System.out.println(risposta);  
  
    // Chiudo ascolto input  
    tastiera.close();  
}
```

2.7 Secondo principio della programmazione – la selezione

In un sistema comune dobbiamo avere la possibilità di gestire il dato in entrata e valutarlo prima di autenticarlo, come una password.

```
// If
if(nome.equals("tomas")) {
    risposta = "benvenuto!";
}
```

Ecco un esempio di **if**, abbiamo:

- **Condizione** = la prima parte, se è vera allora si esegue la seconda parte.
- **Codice da eseguire**.

In questo caso si può fare senza le graffe, in questo modo:

```
// If
if(nome.equals("tomas")) |
    risposta = "benvenuto!";
```

Questo perché c'è solo una riga di codice dopo.

Ci sono diversi tipi di **if**, vediamoli:

2.7.1 – If piatto

```
if(destinazione.equals("londra"))
    costoBiglietto = 120;
if(destinazione.equals("parigi"))
    costoBiglietto = 110;
if(destinazione.equals("oslo"))
    costoBiglietto = 180;
```

Questo tipo di if è detto **if piatto**. Possono esistere contemporaneamente diverse condizioni e il programma le scorrerà una alla volta per verificarne la veridicità. Se una condizione è vera, allora eseguirà il codice ad essa legato, altrimenti passerà oltre.

Ogni condizione viene verificata indipendentemente dalle altre, anche nel caso in cui solo la prima sia vera.

Bisogna risolvere il però un problema, quello del **case sensitive**. Rimando al capitolo 2.1.1.3.

2.7.2 – If else

```
if(eta >= 18)
    risposta = "maggiorenne";
else
    risposta = "NO";

System.out.println(risposta);
```

È un altro esempio di selezione: se la condizione nell'if è vera, allora viene eseguito il codice dell'if. Altrimenti (**else**) viene eseguito il codice dell'else.

Il blocco if else è un blocco unico, un elemento. A ogni if corrisponde un else. Posso avere un if senza else ma non un else senza if.

2.7.3 – If annidati

```
if(eta > 0 && eta < 120)
    if(eta >= 18)
        risposta = "maggiorenne";
    else
        risposta = "minorenne";
else
    risposta = "Hai un'età inferiore a zero oppure superiore a 120";
```

Questo è un esempio di **if annidato**.

2.7.4 – Operatore ternario

```
Scanner tastiera = new Scanner(System.in);
System.out.println("Quanti anni hai?");
int eta = Integer.parseInt(tastiera.nextLine());
```

```
String risposta = eta >= 18 ? "Maggiorenne" : "Minorenne";
```

Un operatore ternario è un modo diverso di scrivere un blocco if-else semplice.

In pratica è come scrivere un blocco if-else in una sola riga:

- String risposta -> significa che dichiaro una variabile di tipo String che si chiama "risposta";
- = -> significa che assegno a risposta il risultato del ternario;
- eta >= 18 -> indica che la condizione corrisponde a if(eta>=18);
- ? -> il punto di domanda indica l'istruzione che il programma esegue nel caso in cui la condizione è vera;
- "Maggiorenne" -> indica il valore che verrà assegnato a risposta se la condizione eta >= 18 è vera;
- "Minorenne" -> indica il valore che verrà attribuito a risposta se la condizione eta >= 18 è falsa.

Si possono creare ternari con più istruzioni ma non è consigliabile. Si possono creare ternari con all'interno altri ternari ma non è consigliabile.

Servono quindi in tutti i casi in cui c'è un blocco di codice if-else che ha una sola istruzione in caso di condizione vera e una sola istruzione in caso di condizione falsa.

```
Scanner tastiera = new Scanner(System.in);
System.out.println("Quanti anni hai?");
int eta = Integer.parseInt(tastiera.nextLine());

//I boolean sono perfetti per i ternari perché possono assegnare
//Di conseguenza o sono true o sono false!
boolean verificaEta = eta >= 18 ? true : false;

String risposta = verificaEta ? "Maggiorenne" : "Minorenne";
```

I boolean sono perfetti per i ternari perché possono assegnare un solo valore alle variabili. Di conseguenza o sono "true" o sono "false".

2.7.5 – Switch

Lo **switch** è un esempio di selezione che ricorda molto gli if piatti ma si comporta in maniera leggermente diversa e introduce nuove possibilità.

```
switch(pizza) {
    case "margherita":
        // qui scrivo il codice da eseguire nel caso in cui
        // il valore di pizza sia margherita
        prezzo = 5;
        break;
    case "marinara":
        prezzo = 4;
        break;
    case "esplosiva":
        prezzo = 10;
        break;
    default:
        prezzo = 15;
        break;
}
```

Lo switch richiede un **parametro**, ossia un valore da inserire all'interno delle parentesi tonde.

Attenzione: lo switch fa una comparazione con quel parametro ma non può eseguire dei calcoli. Non si può scrivere switch(3+1), lo switch non calcola.

Switch(pizza) significa che lo switch compara il valore di pizza con quelli presenti nei vari **case**. Quindi nel caso di "margherita" è come se scrivesse "pizza.equals("margherita")".

Importante è che alla fine di ogni case ci sia un **break**. Ogni case funziona come un programma a sé stante. I case non comunicano tra loro e non possono passarsi i valori o le variabili a un case all'altro.

Va inserito per forza il break perché termina il case e fa uscire il programma dallo switch. Se invece non si mette il break, si va fino alla fine dello switch.

Non è possibile inserire "equalsIgnoreCase()" dentro allo switch. Quindi si risolve con "toLowerCase()" o "toUpperCase()". Vedere capitolo 2.1.1.3.

```

switch(pizza.toLowerCase()) {
case "margherita":
    // qui scrivo il codice da eseguire nel caso in cui
    // il valore di pizza sia margherita
    prezzo = 5;
    break;
case "marinara":
    prezzo = 4;
    break;
case "esplosiva":
    prezzo = 10;
    break;
default:
    prezzo = 15;
    break;
}

```

Conviene metterlo direttamente al parametro, così non modifica il valore della variabile.

2.8 Terzo principio della programmazione – iterazione

Iterare significa ripetere un'istruzione o un blocco di codice finché una condizione è vera.

Le iterazioni in Java prendono anche il nome di **cicli**.

2.8.1 – Ciclo While

```

while(numero > 0 ) {
    System.out.println(numero);
    numero --;
}

```

Si legge: finché il valore di numero è maggiore o uguale a zero, esegui le istruzioni tra le parentesi graffe. La parte tra le parentesi tonde prende il nome di **condizione di iterazione**.

Questo codice viene eseguito se la condizione di iterazione è vera, cioè se il valore di numero è maggiore o uguale a zero.

Se la condizione è vera, il programma stampa il valore di numero. Per evitare un ciclo infinito, è necessario che il valore di numero diminuisca o comunque cambi. In questo caso diciamo che il programma deve diminuire di uno il valore di numero, dopo averlo stampato.

“Numero --” è uguale a scrivere “numero -1”. Lo stesso vale con il “++”;

il ciclo **while** può ripetersi **da 0 volte a infinite volte**.

2.8.2 – Ciclo do while

Il ciclo **while** cicla da zero a infinite volte e verifica subito che la condizione di iterazione sia vera. Se è vera, entra nel ciclo ed esegue il codice, altrimenti no.

```
int numero = 10;

while(numero > 0) {
    System.out.println(numero);
    numero --;
}
```

Diverso è il caso del ciclo **do while**:

```
do {
    System.out.println(numero);
    numero --;
}
while(numero > 10);
```

La differenza è che il ciclo do-while cicla almeno una volta per il principio di **sequenzialità**.

In pratica il ciclo do-while esegue il codice prima di verificare la condizione di iterazione. Se poi vede che la condizione è vera, ripete il ciclo, altrimenti si ferma.

2.8.3 – Ciclo for

```
public static void main(String[] args)
{
    int[] numeri = new int[5];

    numeri[0] = 10;
    numeri[1] = 10;
    numeri[2] = 10;
    numeri[3] = 10;
    numeri[4] = 10;

    int somma = 0;

    for(int i = 0; i < numeri.length; i++) {
        somma += numeri[i];
    }

    System.out.println(somma);
}
```

Questo è un esempio di ciclo for:

- `int i = 0` -> dichiaro e inizializzo una variabile `i`;
- `i < numeri.length` -> condizione di ripetizione, ripeti finché il valore di `i` è minore della grandezza del vettore;
- `i ++` -> incremento, a ogni ciclo aumento di una unità il valore dell'indice.

Il ciclo `for` è specifico per i vettori. Gestisco il contatore, condizione e incremento tutti in solo posto. Inoltre il contatore ha come scope solo il `for`, a differenza del `while`.

```
Scanner tastiera = new Scanner(System.in);
System.out.println("Quanti numeri vuoi considerare?");
int grandezza = Integer.parseInt(tastiera.nextLine());

int[] numeri = new int[grandezza];

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

String numeriPari = "";
String numeriDispari = "";
for(int i = 0; i < numeri.length; i++) {
    if((i % 2) == 0)
        numeriPari += numeri[i];
    else
        numeriDispari += numeri[i];
}

System.out.println(numeriPari + "\n" + numeriDispari);
tastiera.close();
```

Questo è un esempio di come possiamo usare il ciclo `for` per inserire i valori dentro a un vettore.

2.8.4 – Differenze tra diversi cicli

- `Do – while` -> l'unico che ha la condizione di iterazione alla fine, quindi cicla almeno una volta. Utile quando non so quante volte dovrò ciclare, ad esempio quando devo chiedere all'utente di inserire un numero ignoto di libri. Cicla quindi da 1 a infinite volte.
- `While` -> utile quando devo leggere le righe di un file con `hasNextLine()`. Cicla da 0 (nel caso in cui la condizione non venga mai rispettata) a infinite volte.
- `For` -> il ciclo che si usa soprattutto con i vettori. Ha un indice con scope locale e non globale. Indice, condizione e incremento sono raggruppati nello stesso posto. Cicla da 0 a infinite volte. Posso volendo mettere una condizione che non tenga conto dell'indice, ma perderebbe di significato e lo renderebbe più simile a un `while`.

2.9 Leggere file

```

public class Lettura_File_01 {
    public static void main(String[] args) throws FileNotFoundException
    {
        String percorsoFile = "";

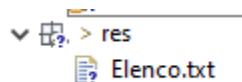
        Scanner file = new Scanner(new File(percorsoFile));

    }
}

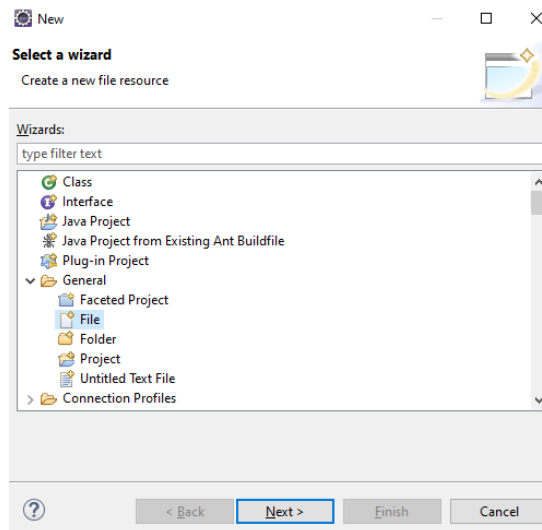
```

Questo è il modo in cui si legge un file, dicendo a Java che se non trova il file deve crashare.

Per prima cosa si crea un package “res” con dentro un file.txt.



Per creare un nuovo file bisogna andare in “new” e selezionare in “general” -> “file”:



Il file sarà così:

```

1 Il signore degli anelli
2 Alien
3 Atto di forza
4 Ghostbusters
5 Ritorno al futuro

```

```

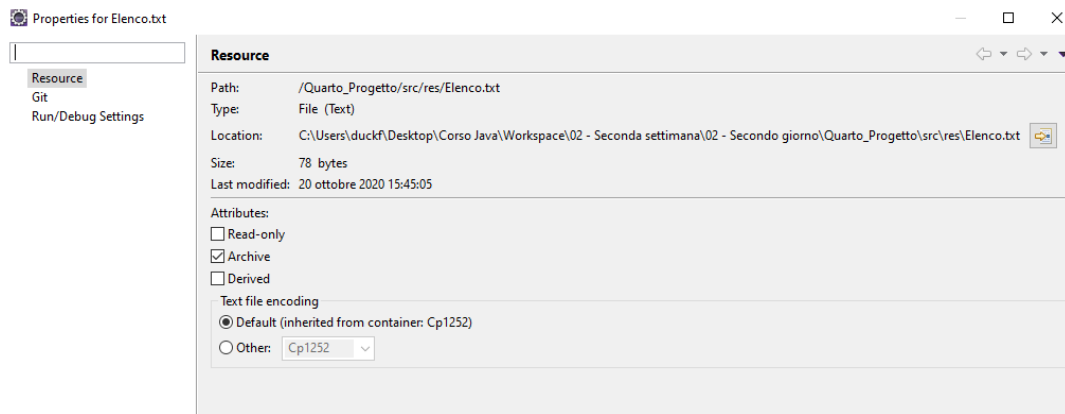
Scanner file = new Scanner(new File(percorsoFile));

```

Con questa riga importiamo lo Scanner e il File.

La variabile “percorsoFile” indica il path del file che vogliamo leggere. Come facciamo a recuperare il path?

Vado sul file che mi interessa, poi faccio tasto destro e clicco su “properties”. Lì devo copiare la “location”:



Dopo aver copiato “location”, incollo in “percorsoFile”, ottenendo questo:

```
String percorsoFile = "C:\\Users\\duckf\\Desktop\\Corso Java\\Workspace\\02 - Seco
```

Cosa importante: devo copiare direttamente all’interno degli apici, perché in questo modo mette in automatico 2 backslash. Se invece copio fuori dagli apici e poi ce li metto attorno, ne fa solo uno, cosa che può dare problemi perché si confonde con l’a capo (\n).

Una volta aperto il file, questo è ciò che facciamo per leggerlo e stampare le righe in console:

```
String percorsoFile = "C:\\Users\\duckf\\Desktop\\Corso Java\\Workspace\\02 - Secondo giorno\\Quarto_Progetto\\src\\res\\Elenco.txt";
Scanner file = new Scanner(new File(percorsoFile));

String riepilogo = "";

while(file.hasNextLine()) {
    riepilogo += "- " + file.nextLine() + "\n";
}

System.out.println(riepilogo);
file.close();
```

Con un ciclo while leggo una riga dopo l’altra e le salvo in una variabile “riepilogo”, in seguito le stampo, una volta finite le linee.

Cosa importante è la condizione del while: **file.hasNextLine()**.

2.10 – I vettori

Un vettore è un insieme ordinato di elementi dello stesso tipo (un vettore di numeri interi può contenere solo numeri interi per esempio).

Un vettore è sempre di tipo **vettore**, ciò che sta al suo interno può essere String, double, int, ecc. ma il vettore sarà sempre solo di tipo vettore.

```
// Dichiaro il vettore
String[] nomi;

// Creo il vettore (inizializzazione)
nomi = new String[3];
```

Questo è un vettore:

- “String” indica il tipo di variabile che può contenere il vettore “nomi”;
- [] è il simbolo che indica il vettore, è di tipo vettore;
- “nomi” è il nome del vettore di tipo vettore che può contenere solo elementi di tipo String.

La prima riga è la dichiarazione del vettore, poi abbiamo la creazione (o inizializzazione) che avviene con la seconda riga:

- “nomi” è il nome del vettore;
- “new” serve per creare il vettore;
- “String[3]” indica il contenuto (String) e la dimensione [3]. [3] indica la dimensione del vettore, ogni vettore è un **insieme finito**. Per poter creare un vettore è sempre necessario specificare la dimensione.

Un vettore può essere vuoto, contenere alcuni elementi, oppure essere pieno. Non è detto che il suo contenuto debba essere sempre pieno o non vuoto.

Un vettore è un insieme **ordinato** di elementi dello stesso tipo. Per ordinare gli elementi al suo interno, ho bisogno di un **indice**. Ogni elemento all’interno del vettore sarà legato al proprio indice. Il valore dell’indice parte da **zero**. Se l’indice parte da zero e il vettore può contenere al massimo 3 elementi, il primo elemento ha indice 0.

Dire che l’ultimo elemento in questo caso ha indice 3 genera un errore in Java. Se a Java chiedete l’elemento con indice 3 è come se steste chiedendo la terza caramella a un sacchetto che ne contiene solo 2.

```
nomi[0] = "Primo nome";
```

- nomi[0] si legge: la posizione zero del vettore nomi di tipo vettore contenente String.
- = “Primo nome” indica il valore della posizione zero del vettore “nomi”.

```
nomi[0] = "Primo nome";

nomi[1] = "Secondo nome";

nomi[2] = "Terzo nome";

System.out.println(nomi[0]);
```

Per stampare il contenuto di una posizione del vettore devo chiamare il numero del vettore e la posizione a cui corrisponde il contenuto che mi interessa.

Scrivere “nomi[0]” oppure “primo nome” è la stessa cosa. Perché “primo nome” è il contenuto di tipo String corrispondente alla posizione zero del vettore (nomi[0]). Quindi se io chiedo a java di stampare nomi[0] lui stamperà il contenuto a quella posizione.

Se dovessi associare un valore a nomi[3] e provare a stamparlo Java genererebbe un errore chiamato:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
at main.Vettori_01.main(Vettori_01.java:19)
```

Che significa che sto chiedendo qualcosa oltre i limiti consentiti dal vettore. Sto chiedendo il quarto elemento in un vettore con solo 3 elementi.

Una volta specificata la dimensione del vettore, resta tale.

```
String nomi[];
nomi = new String[3];

nomi[0] = "Nome 1";
nomi[1] = "Nome 2";
nomi[2] = "Nome 3";

System.out.println(nomi[0]);

nomi[0] = "Pippo";

System.out.println(nomi[0]);|
```

In questo modo riassegno il valore a una delle posizioni del vettore.

```
/ prezzi.length
```

Questo è molto utile perché restituisce la grandezza di un vettore, ossia la sua dimensione.

```
int[] numeri;

numeri = new int[] {1,2,3,4,5,6,7,8,9,10};|
```

Questo è un altro modo per scrivere i vettori, definendo da subito il contenuto. In questo modo, Java capisce da solo la grandezza del vettore. Invece di scrivere la grandezza, scriviamo il contenuto tra le graffe.

La grandezza del vettore viene determinata dagli elementi tra le { }, che in questo caso sono 10.

Inoltre, in questo modo i 10 elementi sono già stati ordinati all'interno del vettore in base a come sono stati presentati.

```

int[] numeri;

String numeriPari = "";
String numeriDispari = "";

numeri = new int[] {1,2,3,4,5,6,7,8,9,10};

for(int i = 0; i < numeri.length; i++) {
    if((i % 2) == 0)
        numeriPari += numeri[i];
    else
        numeriDispari += numeri[i];
}

System.out.println(numeriPari + "\n" + numeriDispari);

```

Questo è un esempio.

Altra cosa importante è che posso inserire i valori direttamente dal for:

```

Scanner tastiera = new Scanner(System.in);
System.out.println("Quanti numeri vuoi considerare?");
int grandezza = Integer.parseInt(tastiera.nextLine());

int[] numeri = new int[grandezza];

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

String numeriPari = "";
String numeriDispari = "";
for(int i = 0; i < numeri.length; i++) {
    if((i % 2) == 0)
        numeriPari += numeri[i];
    else
        numeriDispari += numeri[i];
}

System.out.println(numeriPari + "\n" + numeriDispari);
tastiera.close();

```

Importante in questo caso è il modulo, "%", che serve a restituire il resto di una divisione. È l'unico modo che si ha per verificare se un numero è pari o dispari.