Esercizi Java

Esercizi sulla programmazione orientata agli oggetti

1. Determinare la versione di Java installata nel proprio PC. Creare un semplice programma “hello world” (che stampi a video la stringa “hello world”) utilizzando un editor di testo basilare, tipo *notepad*. Successivamente, compilare e avviare il programma da *prompt* dei comandi di Windows (o da terminale se si utilizza Linux o MacOs).
2. Crea un programma che simula il lancio di due dadi. Il punteggio ottenuto dal giocatore è pari alla somma dei due dadi. Ad ogni lancio l'utente decide se continuare o fermarsi. L'esecuzione del programma termina quando decide l'utente o si raggiunge il punteggio di 100.
   All’uscita del programma vengono stampate a video le seguenti informazioni:
   a. punteggio totale raggiunto dal giocatore
   b. numero di lanci effettuati (ogni singolo lancio consiste nel tirare i due dadi)

   Per generare un numero *pseudo-casuale* puoi utilizzare:
   - il metodo static `Math.random()` , che restituisce un double nell’intervallo *[0, 1[*
     ```
     int rolldice = 1+ (int) Math.random()*6;
     ```
   - il metodo `nextInt(int n)` della classe `java.util.Random`, che restituisce un intero nell’intervallo *[0, n[*
     ```
     Random rnd = new Random();
     int rollagain = 1+ rnd.nextInt(6);
     ```
     Si può anche specificare uno specifico *seed* per la generazione dei numeri, passandolo come parametro al costruttore:
     ```
     Random rnd = new Random(System.currentTimeMillis());
     ```
     Utilizzando sempre lo stesso seed, la sequenza dei numeri generati sarà sempre la stessa.
     Se si omette il parametro nel costruttore, il *seed* verrà cambiato in automatico ad ogni esecuzione, utilizzando come valore il clock di sistema.

   Utilizzare la classe `BufferedReader` per la lettura dell’input:
   ```
   InputStreamReader rd = new InputStreamReader(System.in);
   BufferedReader in = new BufferedReader(rd);
   String ans = in.readLine();
   ```
3. Crea un programma che legge la data di nascita dell’utente come tre input separati (giorno, mese e anno) e fornisce in output il segno zodiacale di appartenenza. Gestire eventuali errori di runtime.
4. Creare una classe `Student`. Ogni oggetto della classe è caratterizzato da **nome**, **cognome**, **data di nascita**, **classe** (es 4C, 3B, 5A, etc.) e **specializzazione** (es. IA, MM, MP, CM, ET, BI). Gli attributi della classe **non devono essere direttamente accessibili dall’esterno**. Fare in modo che non sia possibile inserire dati non validi, ad esempio nome e cognome non possono essere `null` e nemmeno stringhe vuote. Allo stesso modo, l’anno deve essere un numero significativo (ad esempio compreso tra 1970 e 2015. I metodi generano un’**eccezione** nel caso i dati impostati per l’oggetto non siano corretti. Creare un semplice `main` in cui si dimostra il funzionamento della classe. Il `main` deve essere definito in una classe esterna,realizzata in un file a parte (ad esempio `Principale.java`). In particolare:
   a. creare almeno tre versioni del costruttore

b. effettuare un overloading sul metodo per impostare la data di nascita; una versione del metodo riceve un oggetto `LocalDate`, mentre l'altra riceve tre parametri interi (giorno, mese, anno)
c. ridefinire il metodo `toString` in modo che sia possibile stampare a video un oggetto della classe ottenendo tutte le sue informazioni sotto forma di un'unica stringa

Ecco alcune indicazioni e consigli generali:

- la gestione dell'input e dell'output verso l'utente va fatta esclusivamente nel `main` della classe principale: nessun costruttore o metodo della classe `Student` utilizza al suo interno un `BufferedReader` per leggere l'input e nemmeno effettua una stampa a video con `System.out.println()`. L'unico modo corretto per passare dati a un costruttore o a un metodo è attraverso i suo parametri! L'unico modo corretto in cui un metodo restituisce un risultato è attraverso l'istruzione `return`!
- si consiglia inoltre di utilizzare il costrutto `switch-case` per il controllo della specializzazione all'interno del metodo `setSpecializzazione(String spec)`; ad esempio:

```
switch(spec.toUpperCase()) {
        case "IA":
        case "ET":
        case "MM":
        case "MP":
        case "CM":
            this.spec = spec;
            break;
        default:
            throw new Exception(spec+" is not valid!");
        }
```

- è opportuno anche ricordare che in Java il **confronto tra stringhe** viene effettuato con il metodo `equals()` (eventualmente `equalsIgnoreCase()` per ignorare la differenza tra maiuscole e minuscole) e non con l'operatore ==
- la classe `String` possiede anche altri metodi utili come `length()` e `charAt(index)` utilizzabili ad esempio per controllare la correttezza del valore della classe frequentata dallo studente:

```
if(cls!=null && cls.length()==2) {
    char firstChar = cls.charAt(0); // must be a number (1-5)
    char secondChar = cls.charAt(1); // must be a letter
}
```

- in caso si presenti la necessità di trasformare una stringa in intero, utilizzare il metodo `Integer.parseInt(stringa)`, tenendo presente che in caso di conversione non riuscita viene generata un'eccezione di tipo `NumberFormatException`.

5. Modificare l'esercizio precedente:
   a. nel `main`, leggere da tastiera tutte le informazioni da assegnare allo studente, possibilmente su un'unica riga, utilizzando la classe `BufferedReader` o la classe `Scanner`; usare eventualmente il metodo `split()` della classe `String` per suddividere una stringa in base a un carattere separatore

    b. sempre nel `main`, creare un **array** di 5 studenti e fare in modo che l'utente ne inserisca i dati come specificato al punto precedente; gestire in modo ragionevole le situazioni di errore
    c. modificare la classe `Student` sostituendo l'anno di nascita con la **data di nascita completa**, e creare un metodo che restituisca l'età dello studente sulla base della data corrente; utilizzare possibilmente la classe `LocalDate` per la data di nascita e il metodo `Period.between(startDate, endDate).getYears()` per il calcolo dell'età
6. Crea un programma in grado di copiare un file di testo all'interno di un altro file creato *ex novo*.
7. Crea una classe `Punto2D` per rappresentare dei punti sul piano cartesiano. Ogni punto è caratterizzato da un'ascissa e da un'ordinata. Nota: per calcolare la radice quadrata, utilizzare il metodo `Math.sqrt(valore)`. Per la potenza si può utilizzare `Math.pow(base, exp)`. Per convertire una stringa in double si può utilizzare `Double.parseDouble(stringa)`.
    a. Definisci gli **attributi** della classe.
    b. Crea i metodi ***accessor*** (*getter*) e ***mutator*** (*setter*) per consentire l'accesso alle coordinate del punto.
    c. Crea un **costruttore senza parametri** che imposta le coordinate del punto sull'origine degli assi.
    d. Crea un **costruttore con parametri** a cui vengono passate le due coordinate x e y del punto.
    e. Ridefinisci il metodo `toString()`, che restituisce una stringa contenente le coordinate dell'oggetto in formato `"(x , y)"` (al posto di `x` e di `y` ci sono dei numeri.. esempio `"(4 , 1)"`)
    f. Crea un metodo `distanza(Punto2D)`, che restituisce la distanza euclidea tra l'oggetto sul quale viene invocato il metodo e l'oggetto passato come parametro.
    g. Crea un **overloading** del metodo precedente, in modo che i parametri ricevuti siano le due coordinate cartesiane.
    h. Crea un metodo `copia()` per restituire come risultato una copia dell'oggetto sul quale avviene la chiamata.
    i. Crea un metodo `uguale(Punto2D)`, che restituisce `true` se il punto passato come parametro ha le stesse coordinate dell'oggetto su cui viene chiamato il metodo.
    j. Crea i metodi `simmetricoX()`, `simmetricoY()` e `simmetricoO()` che restituiscono l'oggetto `Punto2D` simmetrico rispetto all'asse x, all'asse y e all'origine in riferimento alle coordinate dell'oggetto su cui viene invocato il metodo.
    k. Infine crea una classe `Demo` (contenente solo il metodo `main`) in cui mostrare le funzionalità della classe. In particolare:
        i. l'utente inserisce da tastiera le coordinate di cinque punti sul piano cartesiano (creare un array di oggetti);
        ii. il programma mostra le coordinate dei punti inseriti
        iii. il programma trova il punto più distante dall'origine
        iv. per ogni punto inserito il programma trova quello più distante tra i rimanenti
        v. scelto dall'utente uno dei cinque punti a piacere, vengono istanziati e stampati i suoi simmetrici rispetto agli assi e all'origine
        vi. gestire gli errori di inserimento nel modo più ragionevole
    l. creare la documentazione con Javadoc (crearla all'interno del progetto Eclipse come avviene di default). La documentazione deve riportare le seguenti indormazioni:
        i. autore (nome e cognome dello studente)
        ii. descrizione generale della classe `Punto2D`
        iii. descrizione di tutti i metodi implementati e dei costruttori, specificandone eventuali parametri, eccezioni e, per i metodi, i valori restituiti

8. Crea una classe `Triangolo` per rappresentare l'omonima figura geometrica sul piano cartesiano. In particolare:
    a. Ogni triangolo è caratterizzato da tre vertici, rappresentati con la classe `Punto2D` creata precedentemente (esercizio 2.).
    b. Realizza degli *overloading* del costruttore (quelli che ritieni utili).
    c. Realizza il metodo `getLato(Punto2D a, Punto2D b)`, che restituisce la misura di un lato del triangolo, dati i due vertici.
    d. Utilizzando il metodo `getLato(.., ..)`, realizza i metodi `getPerimetro()` e `getArea()` (quest'ultimo sfrutta la formula di Erone).
    e. Aggiungi i metodi `isEquilatero()`, `isIsoscele()` e `isRettangolo()` , che restituiscono *true* nei rispettivi casi e false altrimenti.
    f. Crea una demo per mostrare le caratteristiche della classe.
9. Crea una classe *Quadrilatero* con lo stesso principio utilizzato per la classe *Triangolo* (in questo caso ci sarà però un vertice in più. Oltre ai metodi `getLato(.., ..)`, `getPerimetro()`, `getArea()` e alla *demo* aggiungi:
    a. Il metodo `isQuadrato()` , che restituisce *true* se il quadrilatero è un quadrato, *false* altrimenti.
    b. Il metodo `isRettangolo()` , che restituisce *true* se il quadrilatero è un rettangolo, *false* altrimenti.
10. Crea una classe *Cerchio* per rappresentare delle circonferenze sul piano cartesiano. Ogni circonferenza è caratterizzata da un raggio e un centro (`Punto2D`, vedi esercizio 2.). Realizza:
    a. Gli *overloading* del costruttore che ritieni opportuni.
    b. I metodi *accessor* e *mutator* necessari.
    c. I metodi `getCirconferenza()` e `getArea()`. (utilizzare il valore di pi-greco fornito da **Math.PI**)
    d. I metodi `isTangente(Cerchio c)` e `isSecante(Cerchio c)`, che restituiscono *true* se l'oggetto è rispettivamente tangente oppure interseca il cerchio passato come parametro al metodo.
    e. Se si tenta di assegnare all'oggetto `Cerchio` un valore negativo del raggio, **viene generata un'eccezione**
    f. Crea una demo per mostrare le caratteristiche della classe e verificare la correttezza dei metodi.
11. Crea una classe `Complex` per rappresentare i numeri complessi. Ricorda che un numero complesso è caratterizzato da una parte reale e una parte immaginaria, quest'ultima moltiplicata per l'unità immaginaria *i* (o *j*), dove $i^2 = -1$. Realizza le seguenti caratteristiche:
    a. Un costruttore con parametri.
    b. I metodi *accessor* e *mutator* per la parte reale e la parte immaginaria.
    c. I metodi `mod()` e `arg()`, che restituiscono rispettivamente il modulo e la fase dell'oggetto.
    d. Il metodo `sum` (somma) con gli *overloading* necessari. Deve essere possibile effettuare la somma sia con un numero reale che con un numero complesso. Il metodo non modifica l'oggetto sul quale viene chiamato, ma restituisce come risultato la somma eseguita (sotto forma di numero complesso).
    e. I metodi `sub` (sottrazione), `mul` (moltiplicazione) e `div` (divisione), con le stesse modalità utilizzate per la somma (vedi punto *d.*).
12. Crea una classe per simulare un distributore di carburante. Per semplificarne il modello, assumi che il distributore disponga solo di un tipo di carburante (es. benzina verde), caratterizzato da un prezzo al litro e da una quantità, intesa come quantità di carburante disponibile nella cisterna del distributore, caratterizzata a sua volta da una capacità totale. Inoltre il distributore deve essere in grado di tenere traccia dell'incasso giornaliero. Realizza i seguenti metodi:

a. `double erogaCarburante(€)`, restituisce i litri erogati al rifornimento

b. `setPrezzo(€)`, imposta il prezzo al litro

c. `rifornisciCisterna(litri)`, rifornisce la cisterna del distributore

d. `getCarburanteRimansto()`, restituisce la quantità di carburante rimasto nella cisterna

e. `getIncasso()`, restituisce l'incasso della giornata

f. `resetIncasso()`, azzera l'incasso (alla fine della giornata)

g. realizza un programma dimostrativo, in cui vengono effettuati dei rifornimenti per diversi importi; mostra la quantità di carburante erogato e quello rimasto nella cisterna; effettua quindi un rifornimento della cisterna e mostra l'incasso della giornata

13. Utilizzando un `ArrayList`, creare un programma in grado di gestire dei dati anagrafici. Ogni record dell'anagrafica contiene nome, cognome, data di nascita, comune di residenza e indirizzo email. La classe per gestire l'anagrafica deve consentire:

    a. l'inserimento di un nuovo record,

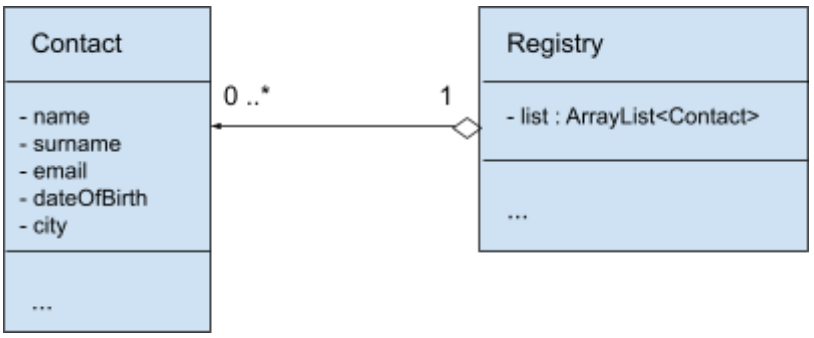
    b. la cancellazione di un record esistente in vari modi (per posizione, per cognome, per email)

    c. la cancellazione dell'intera struttura dati

    d. la restituzione della lista (restituire una copia della lista, altrimenti sarebbe possibile manipolarla in modo diretto dall'esterno, bypassando i metodi della classe Registry!)

    e. la ricerca per nome, cognome, email, anno di nascita

    Creare una demo finale.

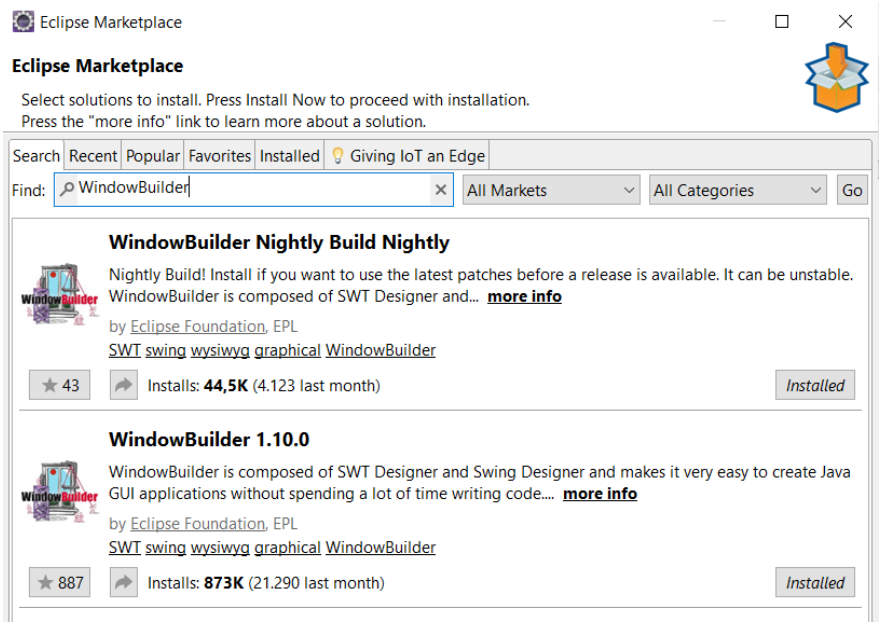


14. *Under construction..*

Esercitazioni su Java Swing

Installazione del plugin WindowBuilder

Installare WindowBuilder utilizzando lo strumento Eclipse Marketplace, che si trova sotto il menu "Help". Cercare il plugin scrivendo "WindowBuilder" nella barra di ricerca:



Metodo di installazione alternativo

Versioni disponibili di *WindowBuilder*:
https://www.eclipse.org/windowbuilder/download.php

Dal menu di Eclipse: "Help" -> "Install New Software"

Incollare questo link nella barra dell'url "Work with" e premere invio (questa versione è stata provata su *Eclipse Mars2* e *Eclipse 2021-03*):
https://download.eclipse.org/windowbuilder/1.9.7/

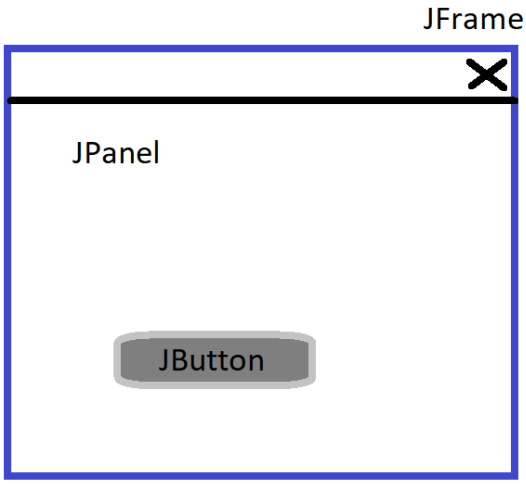
Selezionare dalla *textarea* sottostante i due elementi che compariranno (WindowBuilder e WindowBuilder XWT Support) e premere il pulsante "Next".

Una volta scaricato il plugin, accettare i termini di utilizzo del software (meglio salvare il testo delle condizioni di utilizzo in un file separato in modo da poterlo rileggere in seguito).

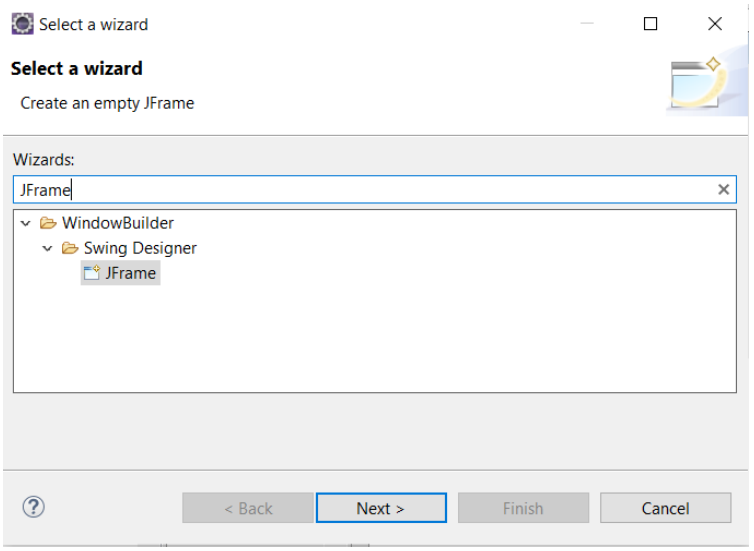
Riavviare Eclipse dopo l'installazione.

Creare un'applicazione Java Swing con Eclipse e WindowBuilder

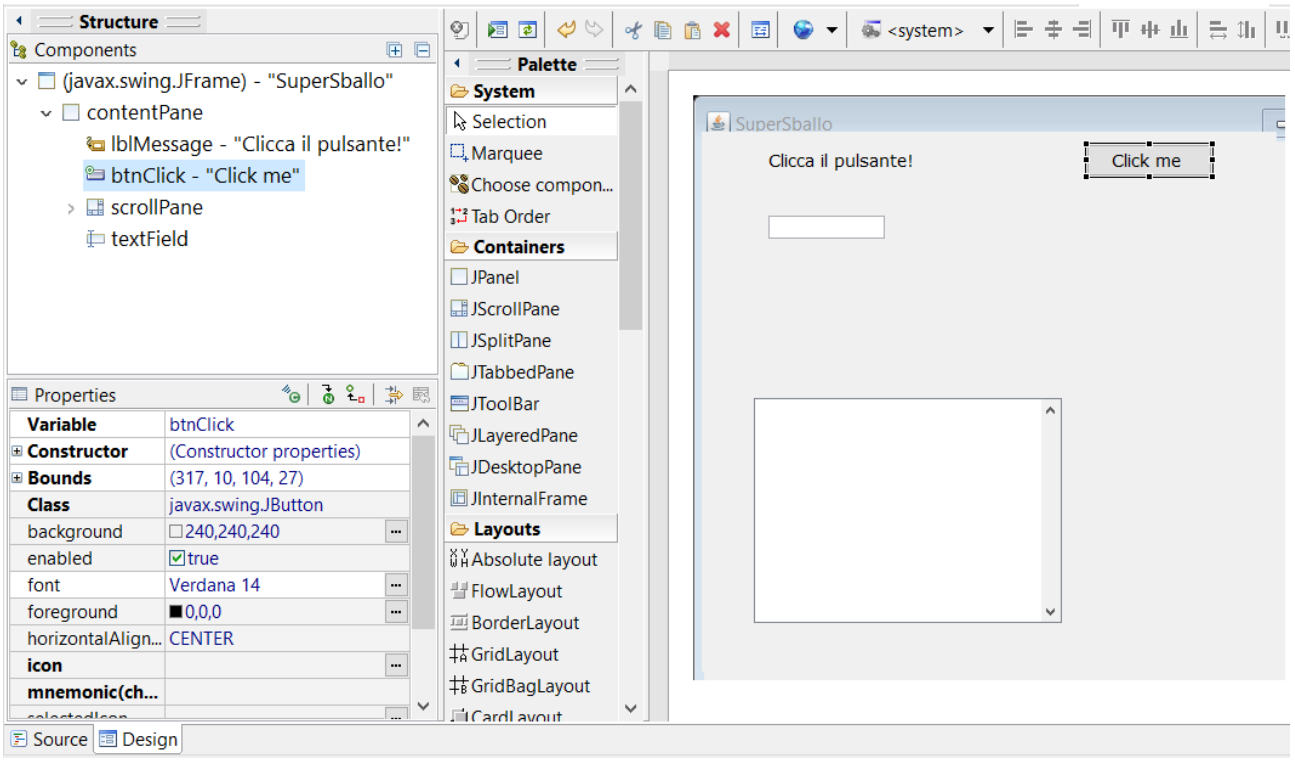
Swing è un framework di Java che permette la creazione di interfacce grafiche. In Swing ogni finestra che funge da interfaccia utente è delimitata da una cornice, rappresentata dalla classe `JFrame`. All'interno della cornice è presente un "pannello" che contiene gli elementi dell'interfaccia utente: questo pannello è realizzato dalla classe `JPanel`. I componenti base dell'interfaccia utente sono molti; i più comuni sono i pulsanti (`JButton`), i campi di inserimento (`JTextField`), le text area (`JTextArea`), le combo box (`JComboBox`), etc.



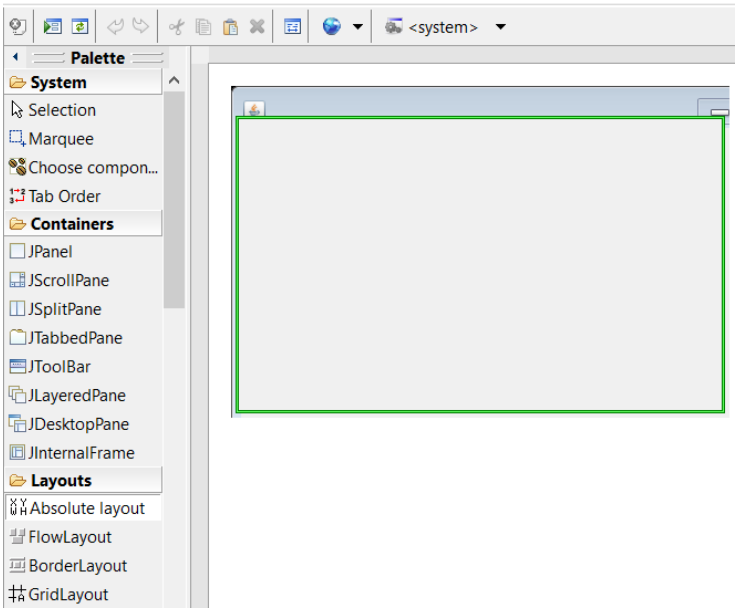
Per creare un `JFrame` all'interno di un progetto Java, selezionare il progetto o il package, cliccare con il tasto destro del mouse -> New -> other. Scrivere "JFrame" sulla barra di ricerca della finestra di dialogo e creare quindi la classe.



Il plugin WindowBuilder consente di realizzare l'interfaccia della finestra in modo grafico, selezionando la linguetta "Design" in basso a sinistra. Ecco come si presenta la modalità "Design":



Prima di iniziare a piazzare i componenti dell'interfaccia grafica nel `contentPane` (`JPanel`) selezionandoli dalla *Palette*, è necessario prima impostare una tipologia di layout per il `contentPane`: per i nostri esempi utilizzeremo sempre l'*absolute layout*, che prevede il posizionamento fisso dei componenti tramite coordinate (x,y). Questo tipo di layout è immediato e intuitivo, ma è poco flessibile, perché i componenti rimangono fissi nella loro posizione anche se la finestra viene ridimensionata. Tuttavia non è al momento nostro scopo quello di creare layout adattivi e responsive.



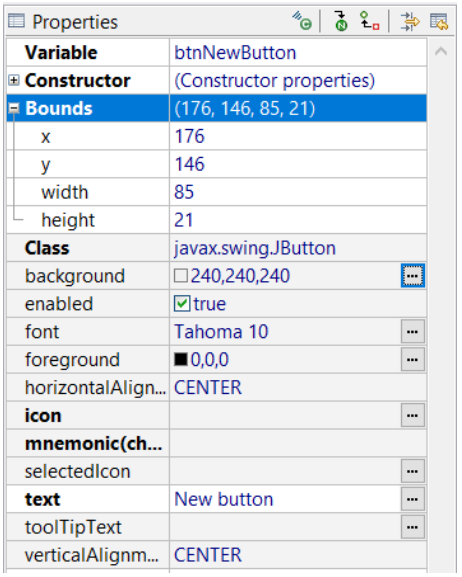
Ogni operazione effettuata sul design sarà tradotta in codice sorgente Java. Ad esempio, aggiungendo un pulsante (`JButton`) alla finestra, verrà automaticamente generato del codice simile al seguente all'interno del costruttore del `JFrame`:

```
JButton btnNewButton = new JButton("New button");
btnNewButton.setBounds(176, 146, 85, 21);
contentPane.add(btnNewButton);
```

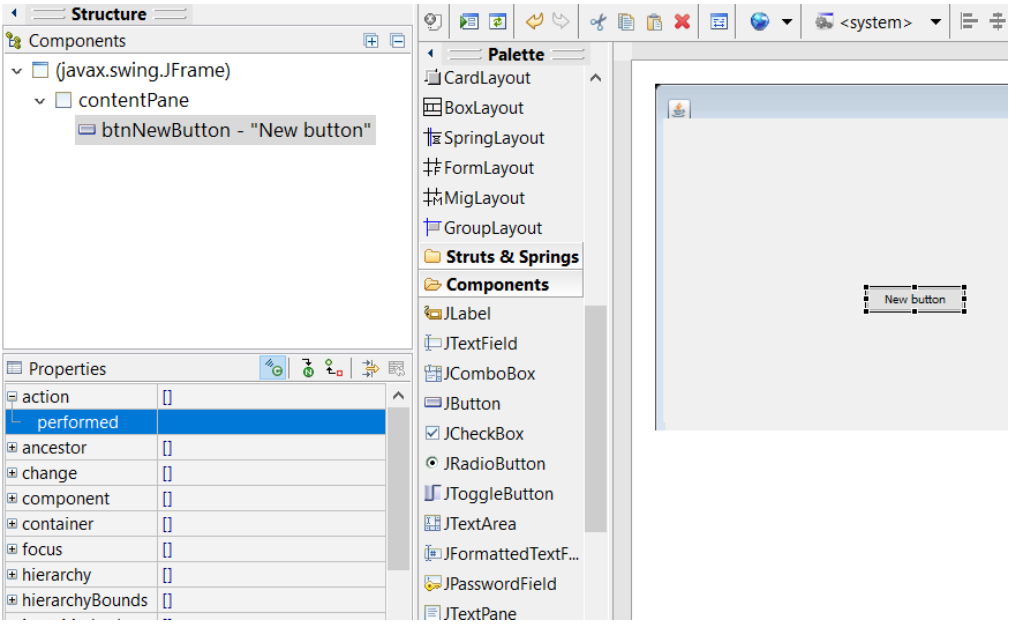
Il significato delle istruzioni è abbastanza intuitivo:

- viene istanziato un oggetto della classe `JButton`, passando al costruttore la stringa `"New button"` (cioè la scritta che poi comparirà sul pulsante)
- vengono impostate le coordinate e le dimensioni del pulsante: i primi due parametri di `setBounds(..)` sono le coordinate (x, y) del pulsante all'interno del riquadro della finestra; il terzo parametro è la larghezza del pulsante, mentre il quarto rappresenta l'altezza (tutto espresso in pixel)
- il pulsante viene aggiunto al `contentPane`, cioè il pannello (`JPanel`) principale della finestra

Tornando alla visualizzazione "Design", si può osservare che molti parametri impostati nel codice si ritrovano nel pannello "Properties" del componente, da cui è possibile facilmente modificare i parametri stessi:



In generale, dopo aver aggiunto i componenti in modalità grafica, sarà necessario tornare alla visualizzazione "Source" del codice sorgente per programmare le risposte agli eventi. Un **evento** viene scatenato in modo **asincrono** (non si sa quando accadrà) come conseguenza dell'azione dell'utente sull'interfaccia grafica (click di un pulsante, selezione di una voce da un menu a tendina, digitazione di un testo all'interno di un campo testuale, ..). Ad esempio, se è presente un pulsante, dovremo specificare quali azioni verranno compiute quando l'utente lo premerà. Nel caso del pulsante ad esempio, per generare l'*event listener* (così viene chiamato in generale il componente software che gestisce la risposta ad un dato evento), si può utilizzare il riquadro "Properties" del componente e cliccare sulla piccola icona verde con la "c" all'interno. A questo punto si può espandere la voce "action" in cima all'elenco degli eventi e cliccare due volte su "performed":

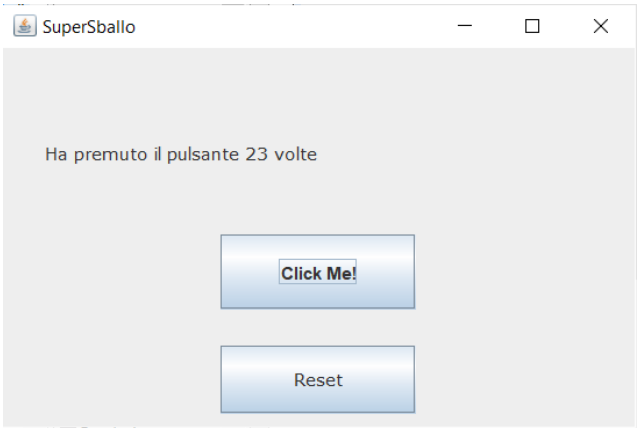


Il codice necessario alla gestione dell'evento "click del pulsante" sarà generato in automatico all'interno del costruttore del `JFrame`. Noi dovremo solo implementare le istruzioni da eseguire all'interno del metodo `actionPerformed()`:

```
btnNewButton.addActionListener(new ActionListener() {
	public void actionPerformed(ActionEvent e) {
		// manage button click here..
	}
});
```

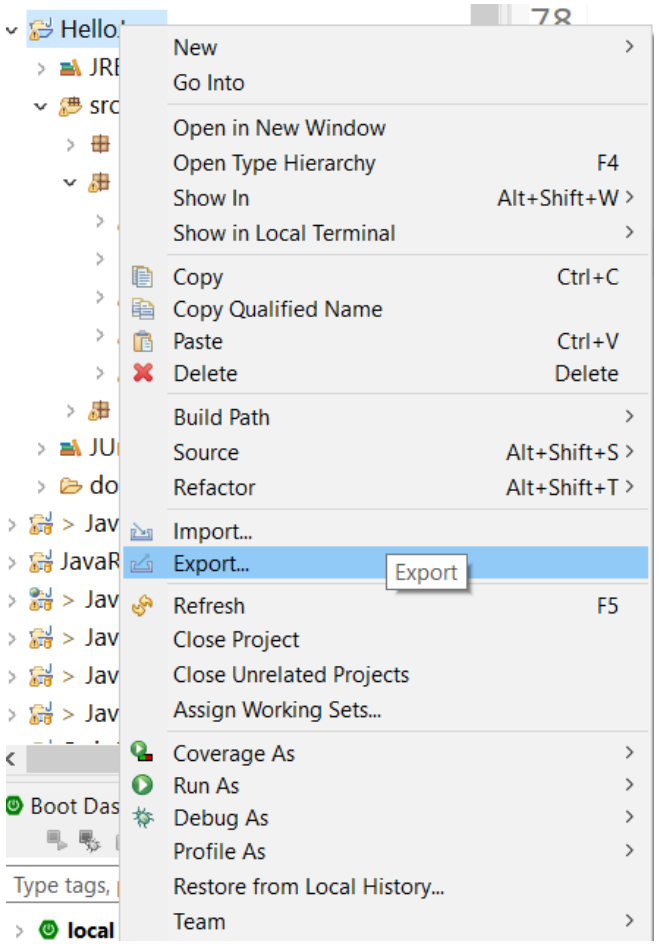
In generale è consigliabile piazzare tutti i gestori di eventi nella parte finale del costruttore del `JFrame`, in modo che non si mescolino con le dichiarazioni dei componenti grafici, che è sempre bene vengano fatte prima.

Realizzare a questo punto una semplice applicazione dimostrativa: una finestra contenente un pulsante e un `JLabel`. Il testo all'interno del `JLabel` deve mostrare il numero di volte che il pulsante è stato premuto fino a quel momento. E' anche presente un pulsante "reset" per reimpostare il contatore dei click:

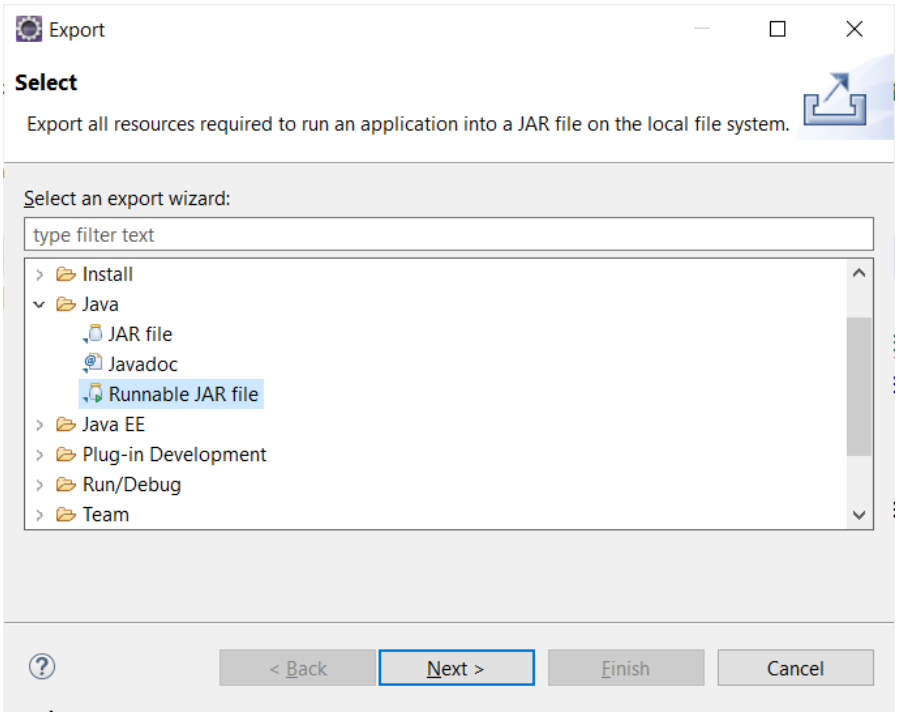


Creazione del Jar eseguibile

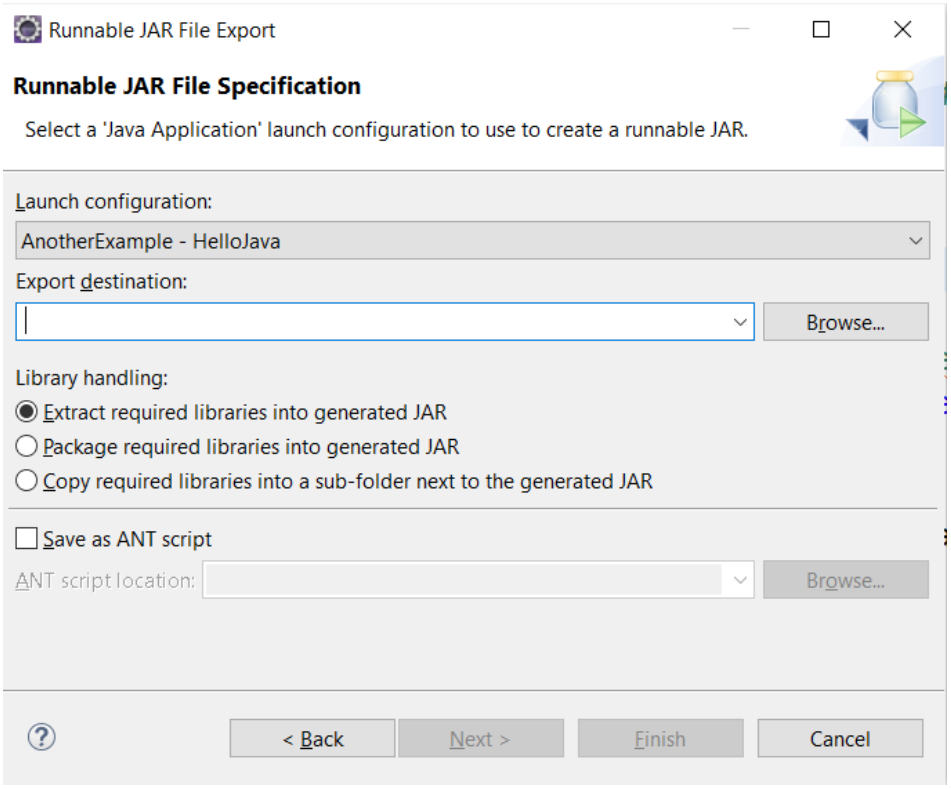
Cliccare su un progetto eclipse con il tasto destro del mouse e selezionare la voce “**Export...**”:



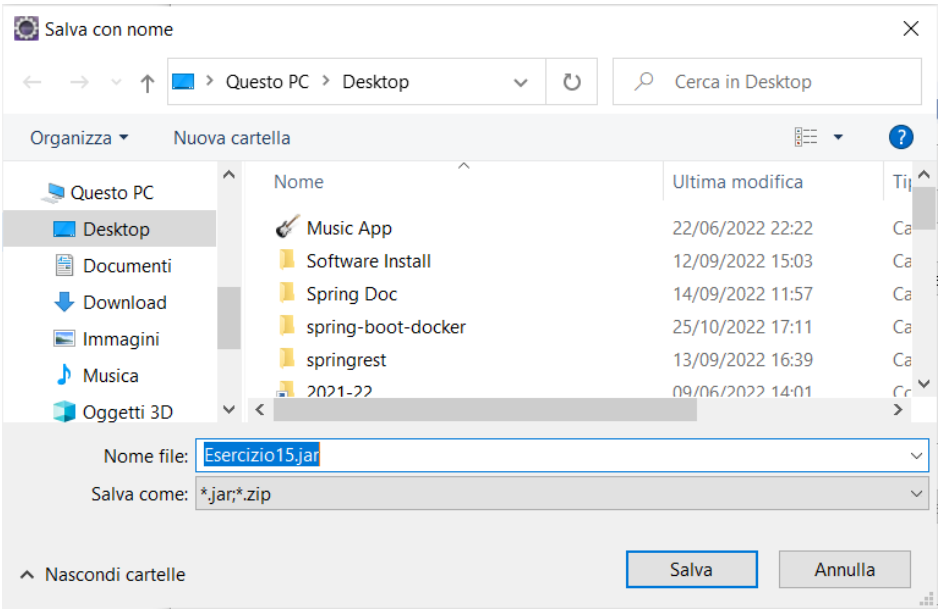
Selezionare “**Runnable Jar file**” dalla successiva finestra di dialogo e cliccare su “**Next**”:



Selezionare l'applicazione da esportare dalla lista di "**Launch configuration**". Poi cliccare su "**Browse…**":



Selezionare la posizione dove salvare il jar e dare un nome al file, scrivendolo nel campo "**Nome file**":



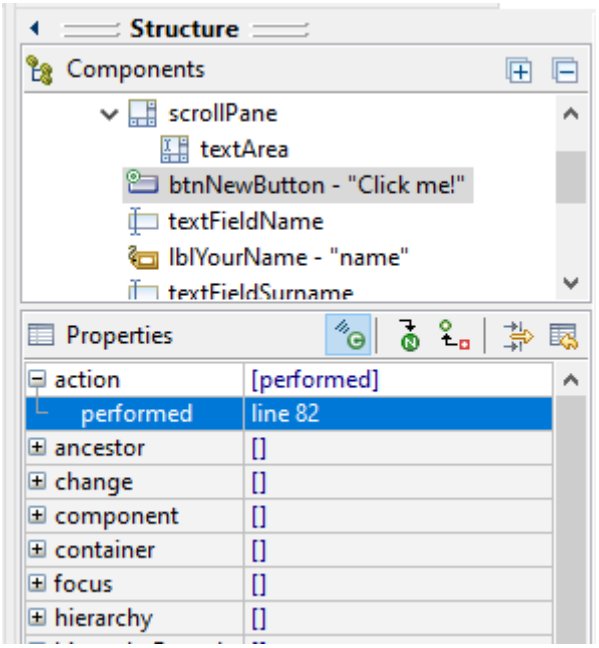
Cliccare su "Salva" e poi su "Finish" per generare il jar eseguibile. Controllare sempre che il jar funzioni effettivamente: si dovrebbe riuscire a lanciare l'applicazione in esecuzione con il doppio click del mouse sul jar stesso.

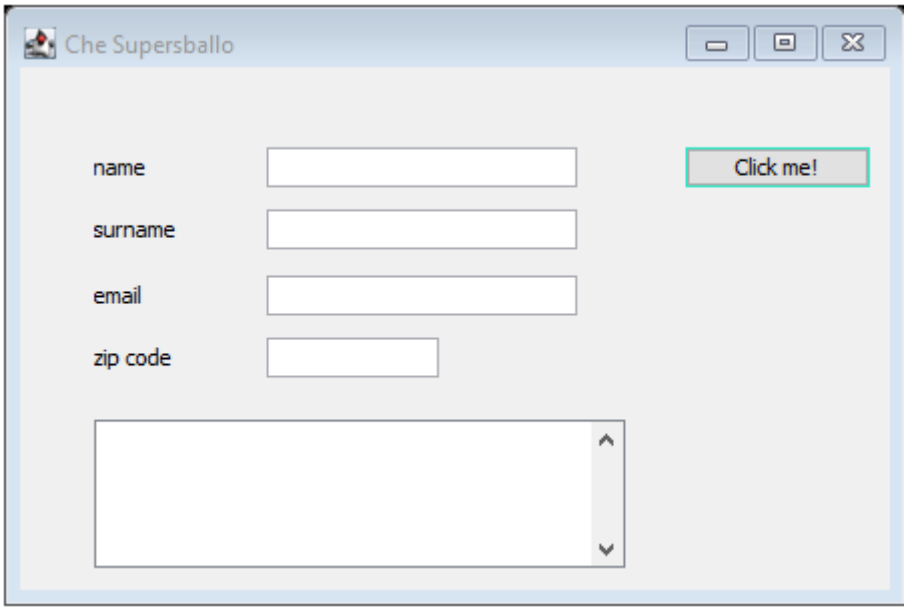
Esercizi

15. Creare un'interfaccia con dei campi testuali come in figura. Quando l'utente preme il pulsante, l'applicazione controlla che nessuno dei campi sia rimasto vuoto. Se è così, scrive il contenuto dei campi all'interno della textArea, ripulendo i campi stessi. Altrimenti avvisa l'utente dei campi non compilati attraverso un *alert dialog*, **elencando esplicitamente i campi mancanti** nel messaggio. Per ottenerlo, utilizzare opportunamente il seguente metodo:

```
JOptionPane.showMessageDialog(MyJFrame.this,
                              "Window Message",
                              "Window Title",
                              JOptionPane.WARNING_MESSAGE);
```

dove `MyJFrame` corrisponde al nome della classe Java da voi creata che rappresenta il `JFrame` della vostra applicazione. Notare che i valori dei campi vengono aggiunti alla textArea attraverso delle "append" (i valori inseriti precedentemente non vengono mai eliminati), per questo è necessario inserire una *scroll bar* verticale nello `JScrollPane` che contiene la `JTextArea`.
Per associare un *event listener* al pulsante, utilizzare il pannello "***Components***" di *WindowBuilder.* Selezionando il pulsante, spostarsi sul pannello "***Properties***" sottostante e cliccare sull'icona "***Show events***", selezionare ed espandere la voce "***action***" e cliccare due volte su "***performed***":





Suggerimenti:
per prelevare il valore da un `JTextField` si utilizza il metodo `getText()`. Per validare il campo si possono utilizzare il seguenti metodi:

- `isEmpty()` - della classe `String` restituisce `true` se la stringa non contiene alcun carattere (stringa vuota, length == 0)
- `isBlank()` - della classe `String` (introdotto da Java 11), restituisce `true` se la stringa non contiene caratteri stampabili (ed esempio quando sono presenti solo degli spazi o dei tab)

16. Integrare l'esercizio precedente con un controllo sui campi **email** e **zipcode**. In particolare, il campo *zipcode* deve essere formato da **5 cifre**, mentre il campo email deve contenere una stringa seguita dal carattere **@**, a sua volta seguito da un'altra stringa, un carattere punto e una stringa finale con minimo due caratteri e massimo quattro. I caratteri validi in un'email sono lettere (a-z, A-Z), numeri, underscore, punto e dash. Per effettuare questo tipo di controllo, utilizzare le espressioni regolari (regex) descritte in seguito.
Regex in Java: per effettuare il confronto tra stringhe, esiste anche il metodo `matches()` a cui è possibile passare come parametro un'espressione regolare, ad esempio:

- `msg.matches("^.*\\bora\\b.*$")` cerca la corrispondenza con una stringa che contenga la parola "ora", ma non come sottostringa; ad esempio, restituirebbe *true* se *msg* fosse una stringa del tipo "dimmi l'ora", oppure "che ora è?", oppure "dammi l'ora attuale"; restituirebbe invece *false* se *msg* fosse "alla buonora!", oppure "Come si chiama tua nuora?"
- `msg.matches("^.*\\btimer\\b.*\\d+.*$")` può essere utilizzata per riconoscere le stringhe che contengono la parola *timer* e il numero che indica i secondi

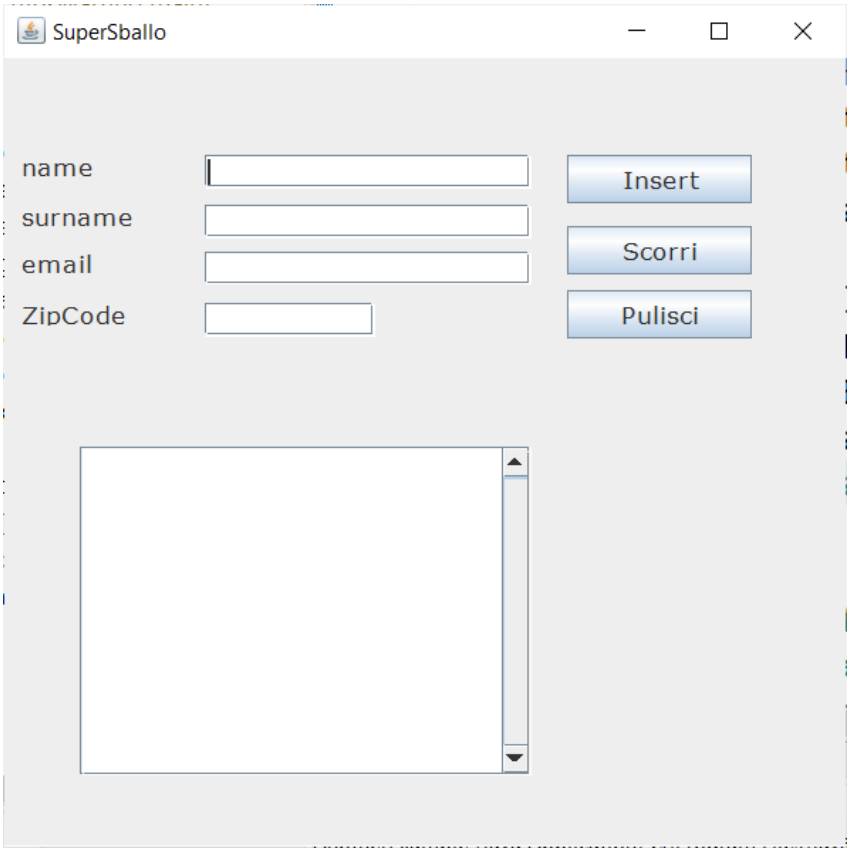
- notare che nelle stringhe Java i ***backslash*** presenti nelle espressioni regolari vanno **preceduti da un altro *backslash***, altrimenti vengono interpretati come sequenze di escape.

  Alcuni esempi sulle espressioni regolari:

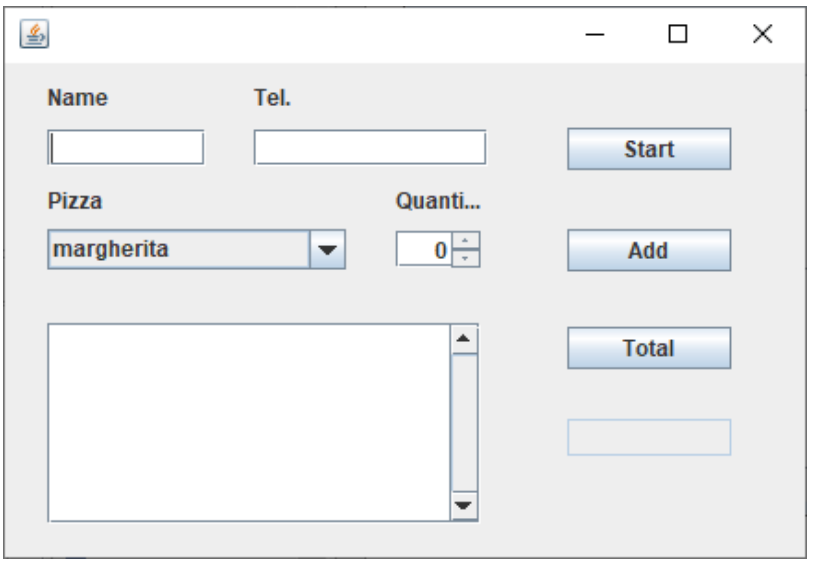
  - ❏ ^ inizio della stringa
  - ❏ $ fine della stringa
  - ❏ * zero o più occorrenze del carattere o dell'espressione precedente
  - ❏ + una o più occorrenze del carattere o dell'espressione precedente

- ? zero o una occorrenza del carattere o dell'espressione precedente
- . carattere qualsiasi
- \. carattere punto
- .+ uno o più caratteri qualsiasi
- .* zero o più caratteri qualsiasi
- \d cifra (carattere compreso tra 0 e 9, equivalente all'espressione [0-9])
- \D carattere non cifra
- \w carattere alfanumerico (equivalente all'espressione [a-zA-Z_0-9])
- \W carattere non alfanumerico
- \s carattere whitespace (spazio, tab, new line)
- \S carattere non whitespace (no spazio, tab o new line)
- \b word boundary, identifica l'inizio e la fine di una parola (esempio \boro\b identifica la parola "oro" da sola, presente ad esempio in "la febbre dell'oro", "oro e argento", ma non in "pandoro e panettone" e nemmeno in "un orologio"
- [a-zA-Z] una lettera qualsiasi, maiuscola o minuscola
- [a-zA-Z]+ una o più lettere qualsiasi, maiuscole o minuscole
- [a-zA-Z]{5} esattamente cinque lettere maiuscole o minuscole
- [a-zA-Z]{5, 10} da cinque a dieci lettere maiuscole o minuscole
- [a-zA-Z0-9] una lettera qualsiasi oppure una cifra (carattere alfanumerico)
- [@ | # | %] carattere chiocciola oppure cancelletto oppure percentuale (| significa or)
- [^a-z] carattere diverso da lettera minuscola (in questo caso ^ significa "non")
- [^@#%] carattere diverso da chiocciola, cancelletto e percentuale
- {n} precisamente n caratteri

17. Modificare l'esercitazione precedente aggiungendo un pulsante "**scorri**". Premendo ripetutamente il pulsante "**scorri**", viene visualizzato di volta in volta un contatto nei campi di inserimento del form secondo l'ordine con il quale i contatti sono stati inseriti. Arrivati alla fine della lista, si ritorna all'elemento iniziale. Aggiungere anche un pulsante "**pulisci**" per ripulire i campi del form (questo pulsante **non** modifica la text area). È consigliabile creare una lista di tipo `ArrayList<Subscription>`, dove `Subscription` è la classe che rappresenta i dati del form. Impedire l'inserimento di una *subscription* già presente nella lista e mostrare, nell'eventualità, un messaggio di errore. Si consiglia di creare il metodo `equals` all'interno della classe `Subscription`, per controllare se due oggetti possiedono gli stessi valori degli attributi. È anche opportuno rimuovere gli eventuali spazi iniziali e finali dai campi del form prima di inserirli nella lista (metodo `trim()`), ed eventualmente sostituire gli spazi multipli con un unico spazio (metodo `replaceAll(regex, string)`, dove il primo parametro è un'espressione regolare che indica cosa si vuole rimpiazzare, mentre il secondo parametro indica la stringa da inserire come rimpiazzo).



18. Creare un'applicazione per ordinare le pizze. L'Interfaccia utente è fatta nel modo seguente:



a. Per poter registrare un ordine è necessario inserire il nome del cliente e il suo numero di telefono, e cliccare quindi sul tasto "Start". Se il nome e il numero di telefono sono stati inseriti correttamente, compaiono scritti nella text area in basso e i rispettivi campi di inserimento vengono ripuliti. Altrimenti compare un Alert che riporta il messaggio di errore. **Controllare che il numero di telefono sia valido! (regex)**
   Il pulsante "Start" rimane poi disabilitato finché non viene premuto il tasto "Total" per fare il conto (requisito extra).
b. Una volta inseriti nome e telefono, si può procedere con l'inserimento dell'ordine delle pizze, che viene effettuato tramite una combo box (scelta della pizza), uno spinner (scelta della quantità) e un pulsante "Add". Premendo quest'ultimo, la/le pizza/e selezionate

vengono aggiunte alla text area con la rispettiva quantità, riportando anche il prezzo unitario. Lo spinner va da 0 a 10, ma la quantità minima per una pizza deve essere 1. Nel caso la quantità sia 0 e si prema il pulsante "Add", visualizzare un Alert di errore. Per impostare le proprietà allo spinner, utilizzare:

```
spinner.setModel(new SpinnerNumberModel(0, 0, 10, 1));
/* init value, min, max, step */
```

Per prelevare il valore dallo spinner (restituisce un oggetto!), utilizzare:
`spinner.getValue();`

Per convertire il valore dello spinner in intero, effettuare un parsing:
`int i = Integer.parseInt(spinner.getValue().toString());`

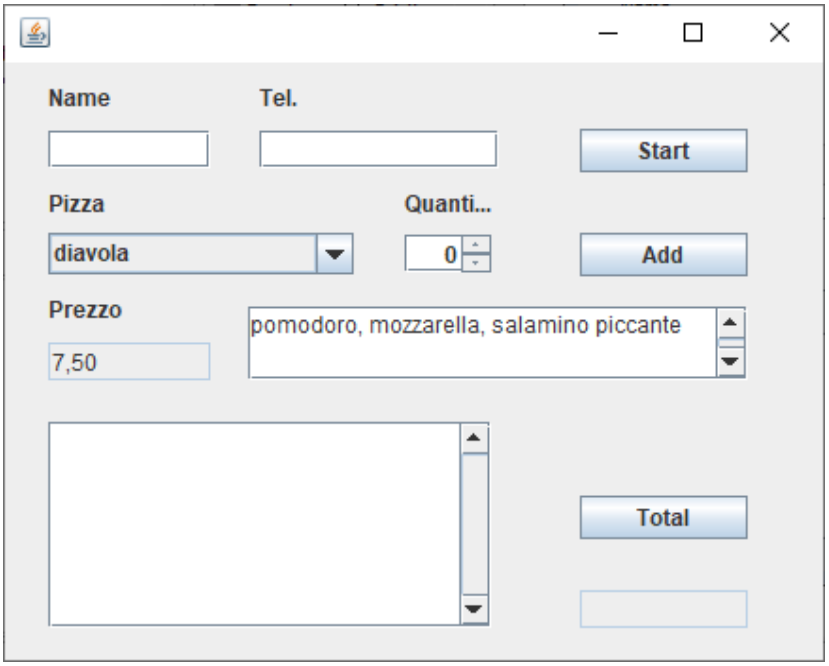
La combo box può essere popolata con il metodo `addItem()`, oppure passandogli l'intero array degli elementi attraverso il costruttore. Si può anche passare un `ArrayList`, purché sia convertito in array classico. Ad esempio:

```
JComboBox<String> combobox
      = new JComboBox<String>(list.toArray(new String[0]));
```

Tenere presente che il parser di WindowBuilder potrebbe non riconoscere correttamente il costruttore con parametro della classe `JComboBox`, quindi l'editor grafico potrebbe non funzionare (per questo è consigliabile aggiungere gli elementi con `addItem()`).
Per ottenere l'elemento selezionato dalla combo box, utilizzare:
`combobox.getSelectedItem();`

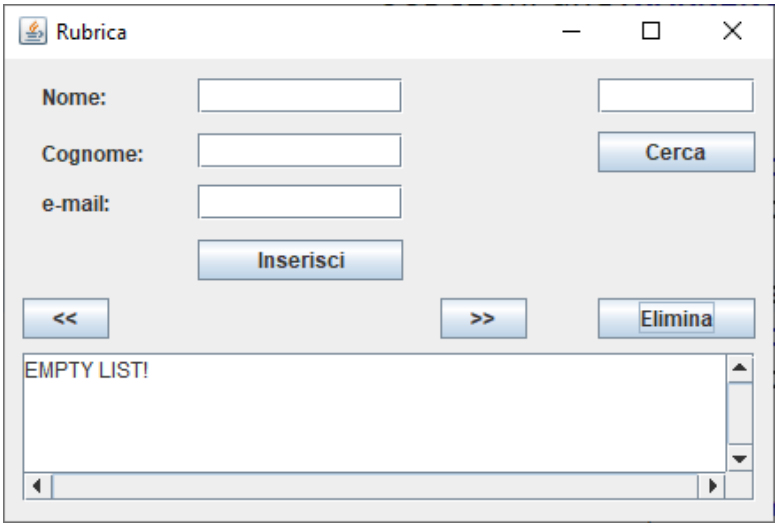
Per rimuovere tutti gli elementi dalla combo box, utilizzare:
`combobox.removeAllItems();`

c. Il pulsante "Total" restituisce il totale visualizzandolo nel textField non editabile posto immediatamente sotto. Dopo aver premuto "Total", per poter effettuare un nuovo ordine è necessario inserire nuovamente il nome cliente e il telefono.
d. Ogni volta che si preme il pulsante "Start" dopo aver inserito nome e telefono validi, l'applicazione viene resettata: la text area viene ripulita e il totale visualizzato è zero se viene premuto il pulsante "Total".
e. **upgrade**: ogni volta che si seleziona una pizza dalla combo box, compaiono in automatico il prezzo e gli ingredienti rispettivamente in un textField e in una text area:



L'**evento della combo box** è lo stesso utilizzato finora per i pulsanti.

19. Crea un'applicazione Java per gestire una rubrica contatti. Ogni contatto è descritto da tre stringhe (nome, cognome e email).



Gestisci la rubrica contatti con un *ArrayList* di oggetti *Contact*, dove la classe *Contact* contiene le informazioni essenziali di un contatto:

```
public class Contact {
	public Contact(String name, String surname, String email) {
		super();
		this.name = name;
		this.surname = surname;
		this.email = email;
	  }
	private String name;
	private String surname;
	private String email;
	public String getName() {
```

```
            return name;
        }

        …
}
```

L'applicazione presenta inoltre le seguenti funzionalità:

- È possibile aggiungere un contatto alla rubrica inserendone i dati nei campi *nome*, *cognome* e *email*, e premendo il tasto "*inserisci*"; ogni inserimento in rubrica viene confermato con un messaggio mediante *alert dialog*, del tipo

  ```
  JOptionPane.showMessageDialog(Anagrafica.this,
                                "messaggio alert..",
                                "titolo alert..",
                                JOptionPane.INFORMATION_MESSAGE);
  ```

  dove *Anagrafica* è il nome della classe derivata da *JFrame*; l'*alert* riporta il dati del contatto inserito e la sua posizione nella lista

- Al momento dell'inserimento viene controllata la validità dei campi del form: nome e cognome devono essere compilati e l'email deve "somigliare" a una mail valida (regex). In caso contrario il contatto non viene inserito e viene mostrato un alert con la descrizione dei dati mancanti o non validi.

- È possibile "navigare" la lista dei contatti mediante i pulsanti *avanti* (">>") e *indietro* ("<<"); il contatto viene visualizzato nella *JTextArea* in basso, che riporterà il nome, il cognome, l'email del contatto e la sua posizione nella lista

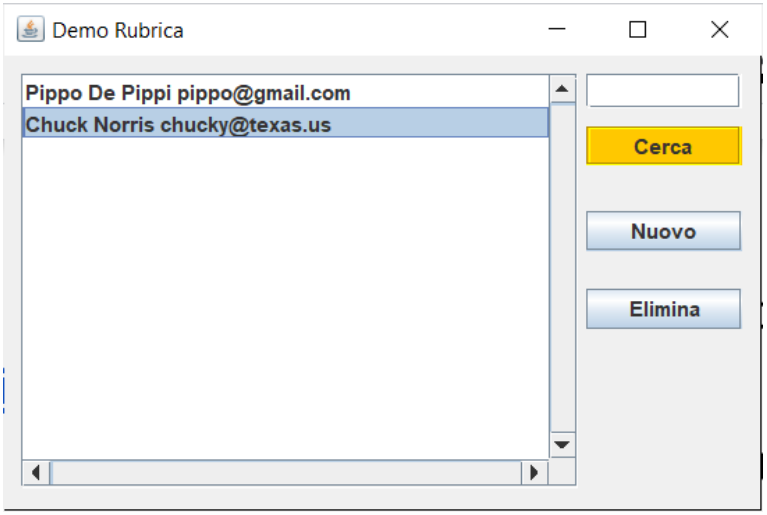
- È possibile eliminare il contatto corrente visualizzato attraverso il tasto "*elimina*"; l'eliminazione del contatto è confermata da un *alert* (*JOptionPane*); dopo l'eliminazione del contatto è necessario anche aggiornare la *JTextArea*, facendo visualizzare il contatto adiacente a quello eliminato, oppure la scritta "EMPTY LIST!" se non ci sono più contatti nella lista

- È possibile effettuare la ricerca per nome/cognome/email di un contatto inserendo la stringa da cercare nel campo ricerca (in alto a destra) e premendo successivamente il pulsante "cerca"; il testo digitato nel campo ricerca viene cercato nei campi *nome*, *cognome* e *email* di ogni contatto e la ricerca è di tipo *case insensitive*; se la ricerca produrrà un risultato, il contatto trovato verrà visualizzato nella *JTextArea*; in caso contrario comparirà un *alert* che avviserà l'utente dell'esito negativo della ricerca. **La ricerca viene effettuata utilizzando le sottostringhe** (vengono trovati tutti i contatti che contengono la sottostringa cercata in uno dei tre campi; utilizzare il metodo `contains()`).

20. Modificare l'applicazione rubrica nel modo seguente:
    a. il form di inserimento non è presente nella finestra principale dell'applicazione, ma compare come finestra secondaria quando viene premuto un pulsante "Nuovo" nella finestra principale.
    b. I contatti non vengono più visualizzati in una `JTextArea`, ma compaiono all'interno di una `JList` o di una `JTable` (a scelta) nella finestra principale. Cercare la documentazione sull'utilizzo di questi due elementi. Sarà poi necessario adattare la funzione "elimina" alla nuova visualizzazione dei contatti.
    c. Inserire un pulsante "Save" per salvare i contatti in un file di testo csv.

d. Inserire un pulsante “Load” per caricare i contatti da un file di testo csv.

21. Modificare l’applicazione a questo link nel modo seguente:



a. ripulire il codice, eliminando l’`ArrayList` non necessario e utilizzando solo il `DefaultListModel`. Notare che il componente `JList` è associato all’oggetto `DefaultListModel`, che funge da contenitore per gli elementi che compaiono nella *user interface*:

```
JList listView = new JList();
DefaultListModel<Contact> dlm = new DefaultListModel<Contact>();
listView.setModel(dlm);
```

I dati dei contatti possono essere aggiunti ed eliminati dal `DefaultListModel` in modo simile a come viene fatto per un `ArrayList`:

```
dlm.addElement(new Contact(..));
dlm.remove(index);
dlm.clear();
```

b. aggiungere data di nascita, indirizzo, comune di residenza e numero di telefono ai dati dei contatti (controllare che la data, l’email e il numero di telefono siano corretti)

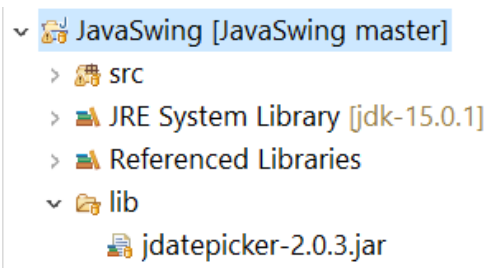
c. permettere la cancellazione dell’intera lista contatti con apposito pulsante ‘elimina tutto’

d. (facoltativo) utilizzare la classe `JDatePicker` per le date (scaricare il jar qui):

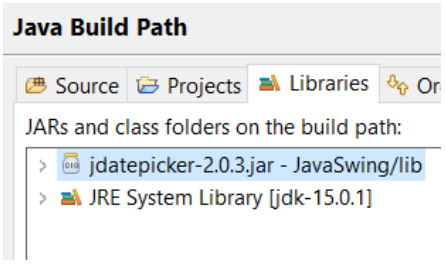
```
import org.jdatepicker.*;
…
JDatePicker datePicker = new JDatePicker(new Date());
datePicker.setBounds(20, 139, 104, 22); // set widget position
contentPane.add(datePicker);  // add widget to content pane
// get selected date:
Date selectedDate = (Date) datePicker.getModel().getValue();
```

Aggiungere il jar al progetto:

- da Eclipse, creare una nuova cartella “lib” e trascinarci all’interno il jar



- aprire le proprietà del progetto e aggiungere la libreria jar alle librerie con il pulsante “ADD JARs..”



e. la ricerca mostra i risultati direttamente all’interno della `JList`; se non viene inserita alcuna stringa nel campo di ricerca, vengono visualizzati tutti i contatti presenti; si consiglia di utilizzare un secondo oggetto `DefaultListModel` solo per memorizzare i risultati della ricerca..

f. fare in modo che la ricerca venga effettuata senza dover premere il pulsante “Cerca”, ma semplicemente digitando qualcosa all’interno del campo di ricerca; per far questo si utilizza l’interfaccia `DocumentListener`, implementando i tre metodi relativi rispettivamente alla modifica del contenuto del campo testuale, alla cancellazione di un carattere e all’inserimento di un nuovo carattere:

```
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
…
searchField.getDocument().addDocumentListener(new
DocumentListener() {
      public void changedUpdate(DocumentEvent e) {
      }
      public void removeUpdate(DocumentEvent e) {
      }
      public void insertUpdate(DocumentEvent e) {
      }
});
```

g. gli elementi della `JList` sono cliccabili e mostrano, in qualche modo, i dettagli del contatto selezionato. Ecco un esempio di *action listener* che rileva il doppio *click* del mouse su un elemento della lista e ne ricava l’indice:

```
jlist.addMouseListener(new MouseAdapter() {
    public void mouseClicked(MouseEvent evt) {
        JList list = (JList) evt.getSource();
        if (evt.getClickCount() == 2) {
            // Double-click detected
            int index = list.locationToIndex(evt.getPoint());
        } else if (evt.getClickCount() == 3) {
            // Triple-click detected
            int index = list.locationToIndex(evt.getPoint());
        }
    }
});
```

22. Under construction..

Esercizi di recupero

1. Creare una classe per rappresentare le date, simile alla classe `LocalDate`. La classe premette, attraverso il suo costruttore, di impostare giorno, mese e anno direttamente nel momento in cui l'oggetto viene istanziato. In qualche modo è anche necessario impedire a chi utilizza la classe di creare date con valori inconsistenti, tipo 31-11-2020 (novembre ha 30 giorni), oppure 29-02-2021 (il 2021 non è un anno bisestile *(*)*). nel caso i dati passati dall'utilizzatore della classe fossero errati, viene generata un'eccezione. Inoltre, attraverso opportuni metodi della classe, è possibile apportare modifiche all'oggetto (ad esempio impostare il giorno, il mese, o l'anno), sempre controllando la consistenza dei dati ad ogni modifica. Prevedere infine i metodi getter per ogni attributo della classe, e il metodo toString che restituisce la data il formato stringa "dd/mm/yyyy".
   (*) un anno è bisestile se è perfettamente divisibile per 400, oppure se è perfettamente divisibile per 4 ma non per 100. Creare all'interno della classe "Data" un main e scrivere un breve programma dimostrativo delle funzionalità della classe.
2. In riferimento all'esercizio precedente, realizzare dei metodi per effettuare somme e sottrazioni sugli oggetti "Data".
3. Crea un convertitore di valute. I valori di cambio sono definiti nella classe che effettua le conversioni e non sono modificabili, ma possono essere conosciuti. Dovrebbe essere possibile effettuare le seguenti conversioni:
   a. da Euro a Dollaro USA e viceversa
   b. da Euro a Yen e viceversa
   c. da Euro a Sterlina e viceversa
   d. da Euro a Yuan e viceversa
   e. da Euro a Rubli e viceversa

   Specifica in che modo conviene definire i metodi e valori di cambio a livello di linguaggio Java.

4. Crea una classe per rappresentare luoghi sotto forma di coordinate lat/lon. Prevedi un costruttore con due parametri (lat e lon) che genera un'eccezione se i valori passati non sono accettabili. In particolare:
   a. il valore della latitudine deve essere compreso tra -90° e +90°, mentre il valore della longitudine è tra -180° e +180°
   b. prevedere la possibilità di effettuare le operazioni di somma sia sulla latitudine che sulla longitudine. I risultati delle operazioni devono sempre essere compresi negli intervalli precedentemente specificati
   c. il metodo toString() restituisce i valori di lat e lon del punto
   d. il metodo distance(..) restituisce la distanza in km da un'altra coppia di coordinate, calcolandola con la formula di *haversine*
5. *under construction*