Parte 16

GUI – Parte seconda

[C.M.Coolidge – A Friend in Need, 1903]

Obiettivo

- Realizzeremo la semplice interfaccia grafica del nostro programma di gestione delle sequenze
  - Per raggiungere questo obiettivo analizzeremo tutti i **widget** che ci serviranno
- Vedremo brevemente i principi e le linee guida da seguire per realizzare **GUI professionali**

Aggiunta GUI a Gseq (1)

Per fornire il nostro programma di gestione delle sequenze di interfaccia grafica possiamo adottare il *seguente approccio*

- Partiamo dai sorgenti di *progetto_multifile*
  - *Zip in cartella compilazione*
- Lasciamo inalterati tutti i moduli tranne il **Main**
- Attualmente il **Main** interagisce con l'utente mediante un **menù testuale**
- Dobbiamo modificarlo per gestire una **più complessa interfaccia grafica**

Aggiunta GUI a Gseq (2)

Nella funzione **main** lasciamo solo:

- La definizione delle variabili che ci servono
  - Probabilmente poi per semplicità definiremo la sequenza come variabile globale, ma vedremo dopo
- La lettura delle opzioni da riga di comando
- L'inizializzazione della struttura dati

Aggiunta GUI a Gseq (3)

- Eliminiamo invece la parte di gestione del menù testuale, perché quest'ultimo sarà **sostituito dai menù della GUI**
- Per assicurarci che non ci siano problemi, fatti i tagli **ricompiliamo** questa versione del codice ed eseguiamola
  - Ovviamente il programma dovrà terminare immediatamente dopo essere stato invocato

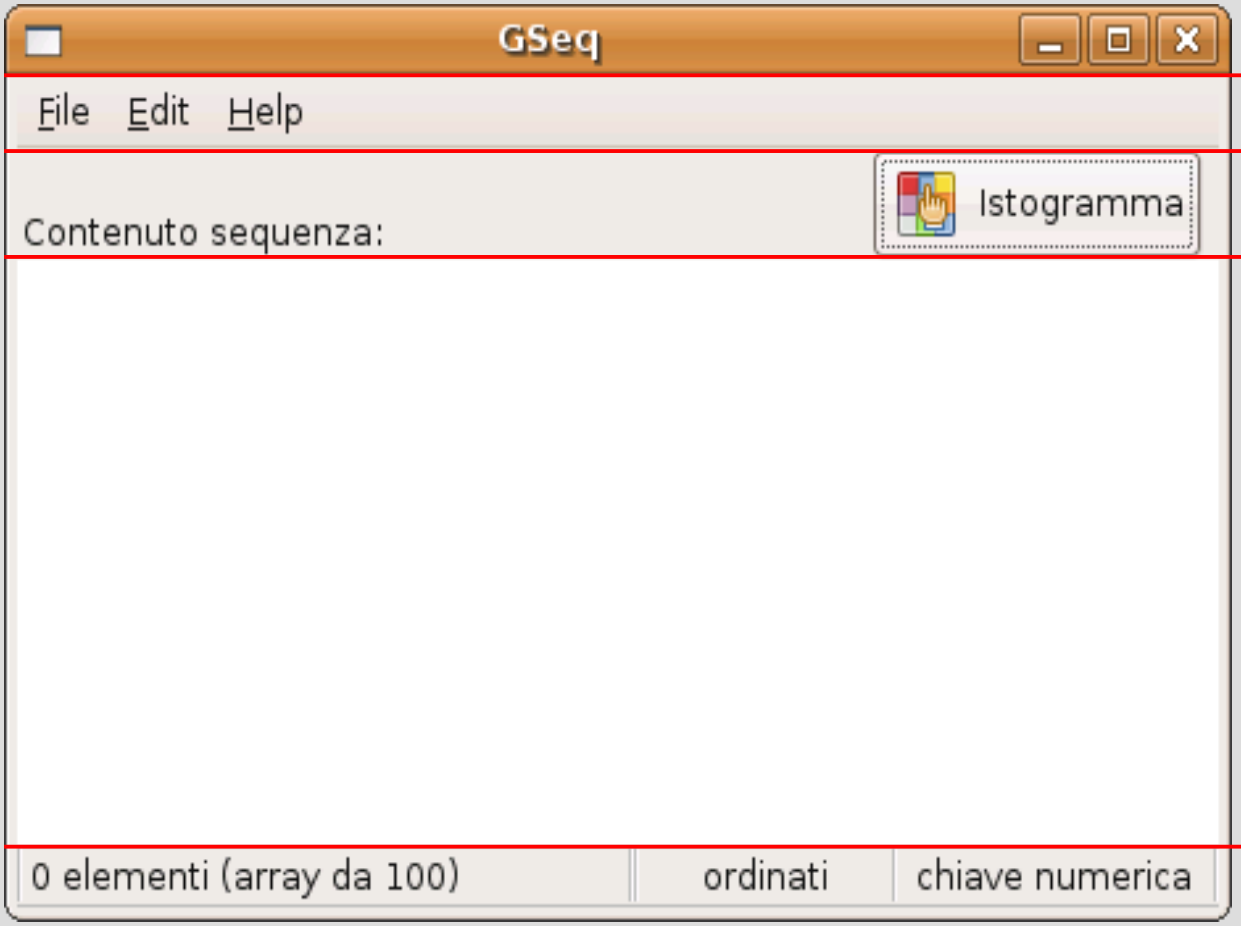
Aggiunta GUI a Gseq (4)

- Come programma di riferimento durante la scrittura della **GUI** utilizzeremo i sorgenti in *GUI_dev/progetto_GUI*
- Tali sorgenti non contengono l'intero programma **GSeq** in versione grafica, ma solo ciò che faremo in questa parte
- Praticamente i due file che cambiano in modo significativo rispetto alla versione non grafica del programma sono **GSeq.cc** ed il **Makefile**, come stiamo per vedere

Disegno finestra principale

- Iniziamo quindi a disegnare la finestra principale con **Glade-3**
- A questo scopo, cominciamo dall'individuare gli **elementi principali** da cui vogliamo sia costituita l'interfaccia grafica, nonché la loro **collocazione**

Scomposizione (1)



Barra dei menù

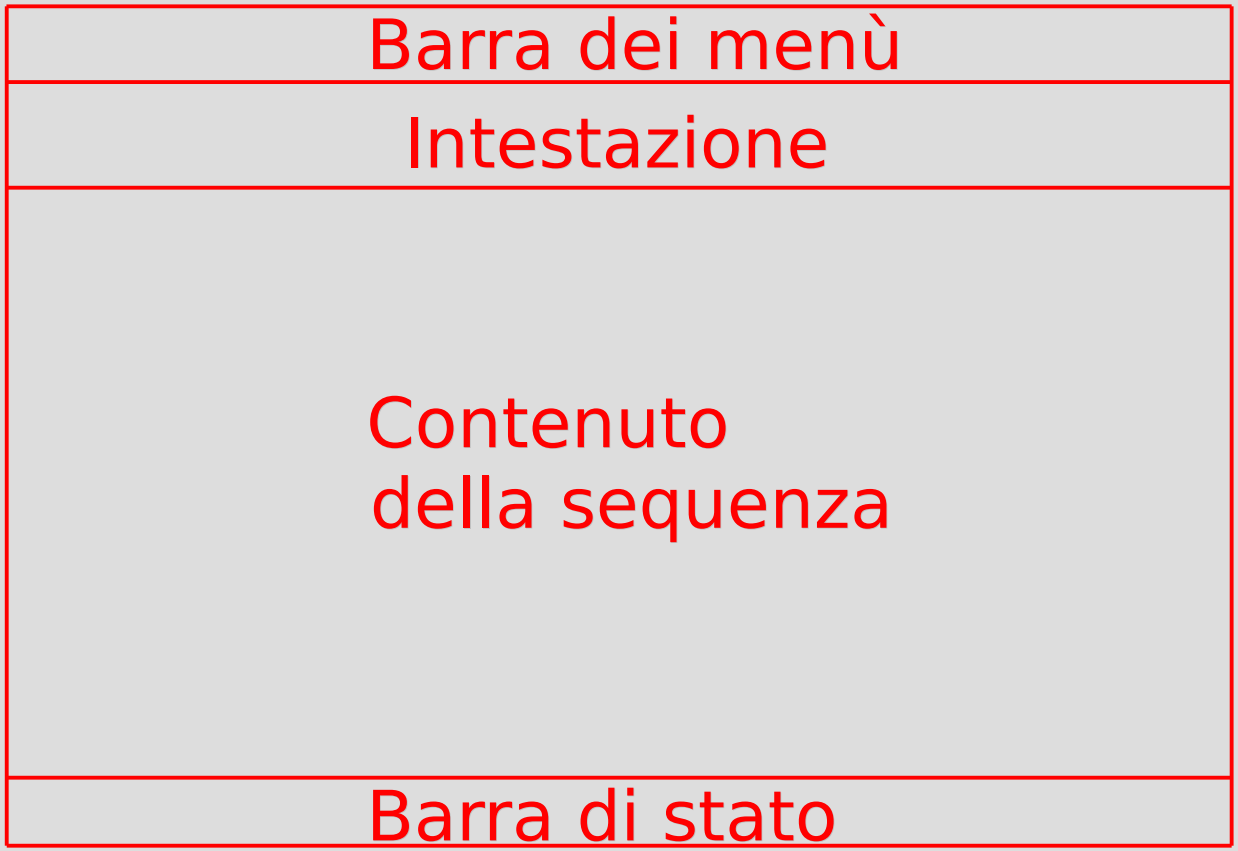
Intestazione

Contenuto della sequenza

Barra di stato

Scomposizione (2)

- Immaginando di farle una radiografia, la finestra è costituita quindi dalle seguenti parti:



Collocazione dei Widget

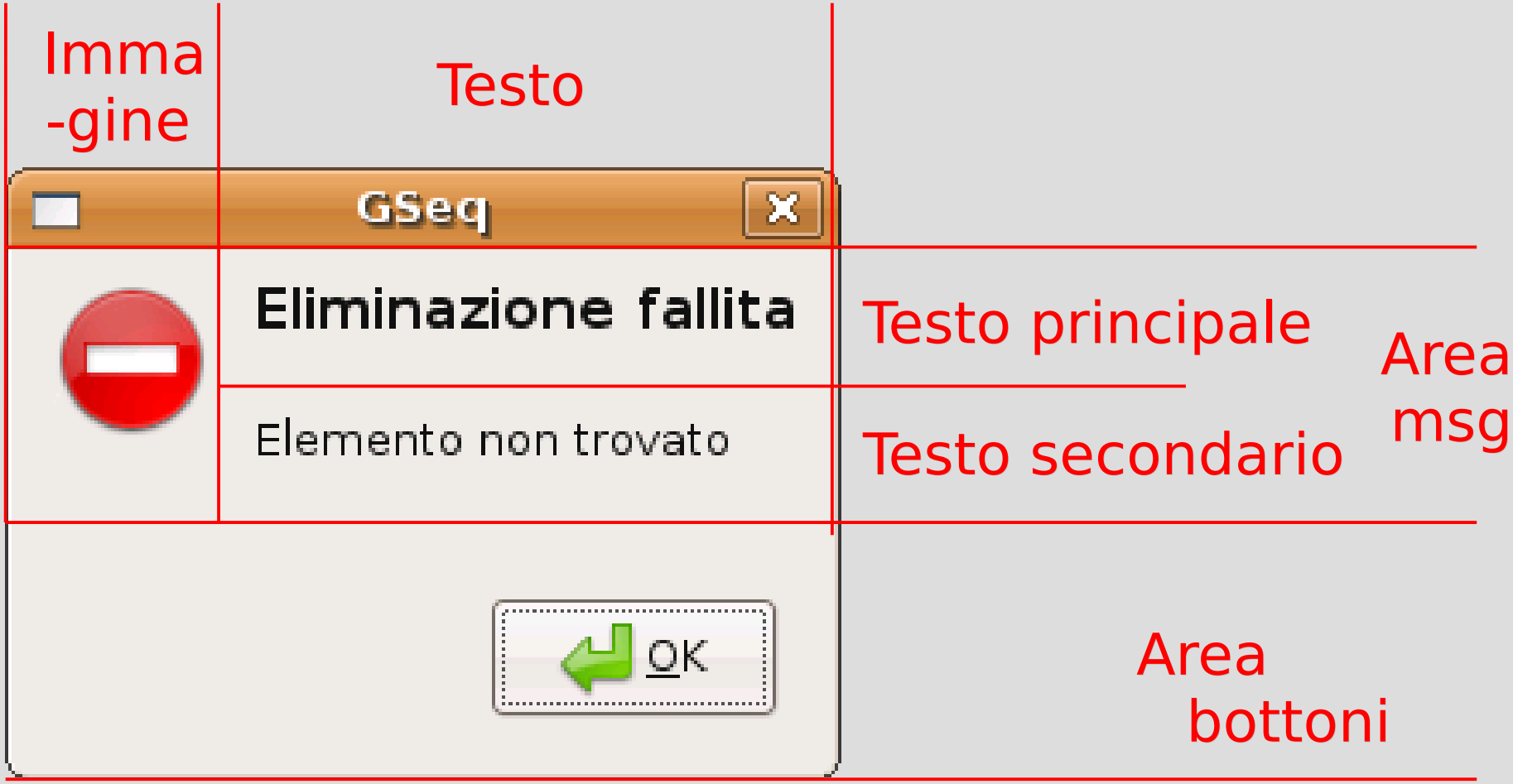
- Abbiamo quindi individuato 4 parti di base, disposte l'una sotto l'altra
- Prima di procedere con la collocazione dei corrispondenti **widget**, introduciamo il concetto di collocazione di **widget figli** all'interno di un **widget contenitore**
- A tale scopo, come ulteriore esempio, proviamo ad individuare anche le parti di cui è composta una **finestra di dialogo** che si apre ad esempio quando fallisce l'eliminazione di una sequenza

Scomposizione (1)

- Si possono individuare due aree principali:

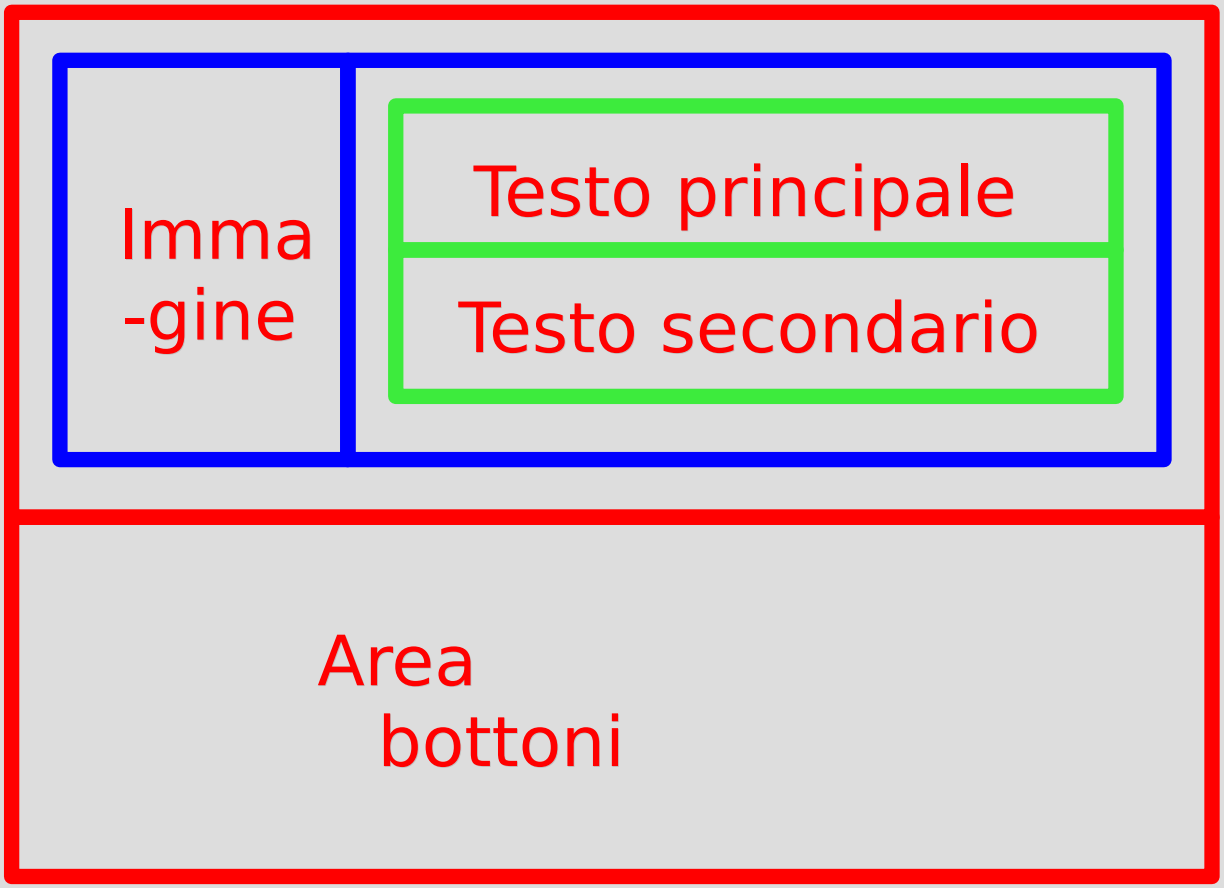
Scomposizione (2)

- L'area messaggi può ulteriormente essere scomposta come segue:



Scomposizione (3)

- Lo schema può essere visto come:



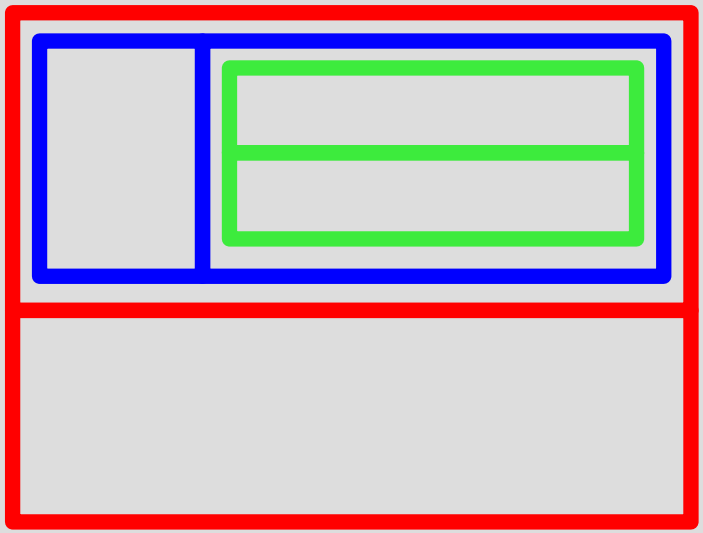
Scomposizione (4)

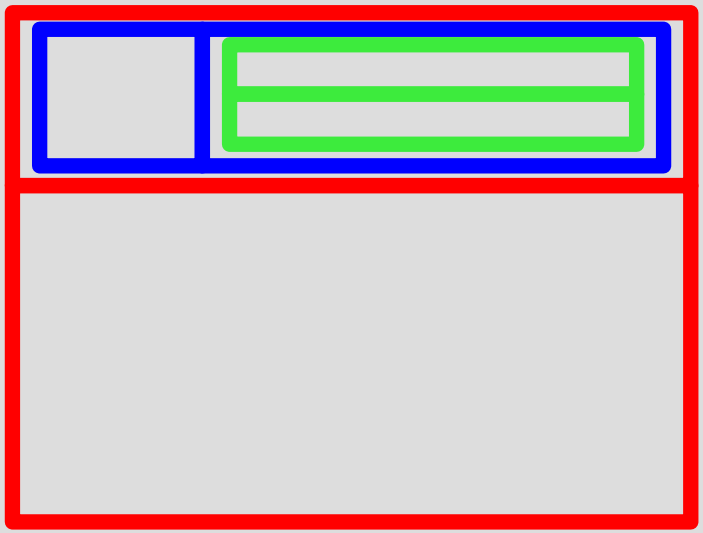
- Due **box principali rossi** disposti **verticalmente**
  - Il box rosso superiore contiene **due box blu** disposti **orizzontalmente**
    - Il box blu di destra contiene **due box verdi** disposti **verticalmente**

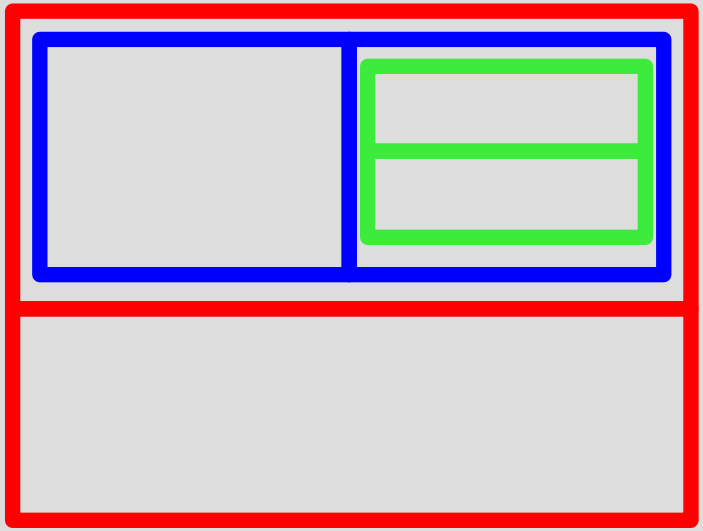
Equivalenza (1)

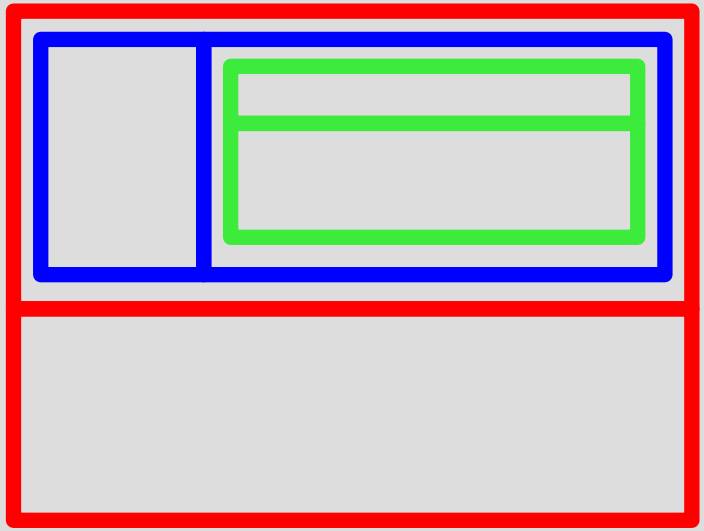
- In termini di determinare quali box sono contenuti in altri box e di posizione relativa dei box, lo **schema di scomposizione è ovviamente indipendente dalle dimensioni dei singoli box**
- Ad esempio le seguenti scomposizioni sono tutte **equivalenti**

Equivalenza (2)









Horizontal e Vertical box (1)

- La libreria **GTK+** permette di collocare i **widget figli** nel **widget padre** in base ad una logica di scomposizione in box
- I **widget** contenitore più semplici che permettono di inserire almeno due elementi in un **widget padre** sono gli **horizontal box e i vertical box**

Horizontal e Vertical box (2)

- **Horizontal Box**

  Sequenza di box consecutivi disposti l'uno dopo l'altro in orizzontale

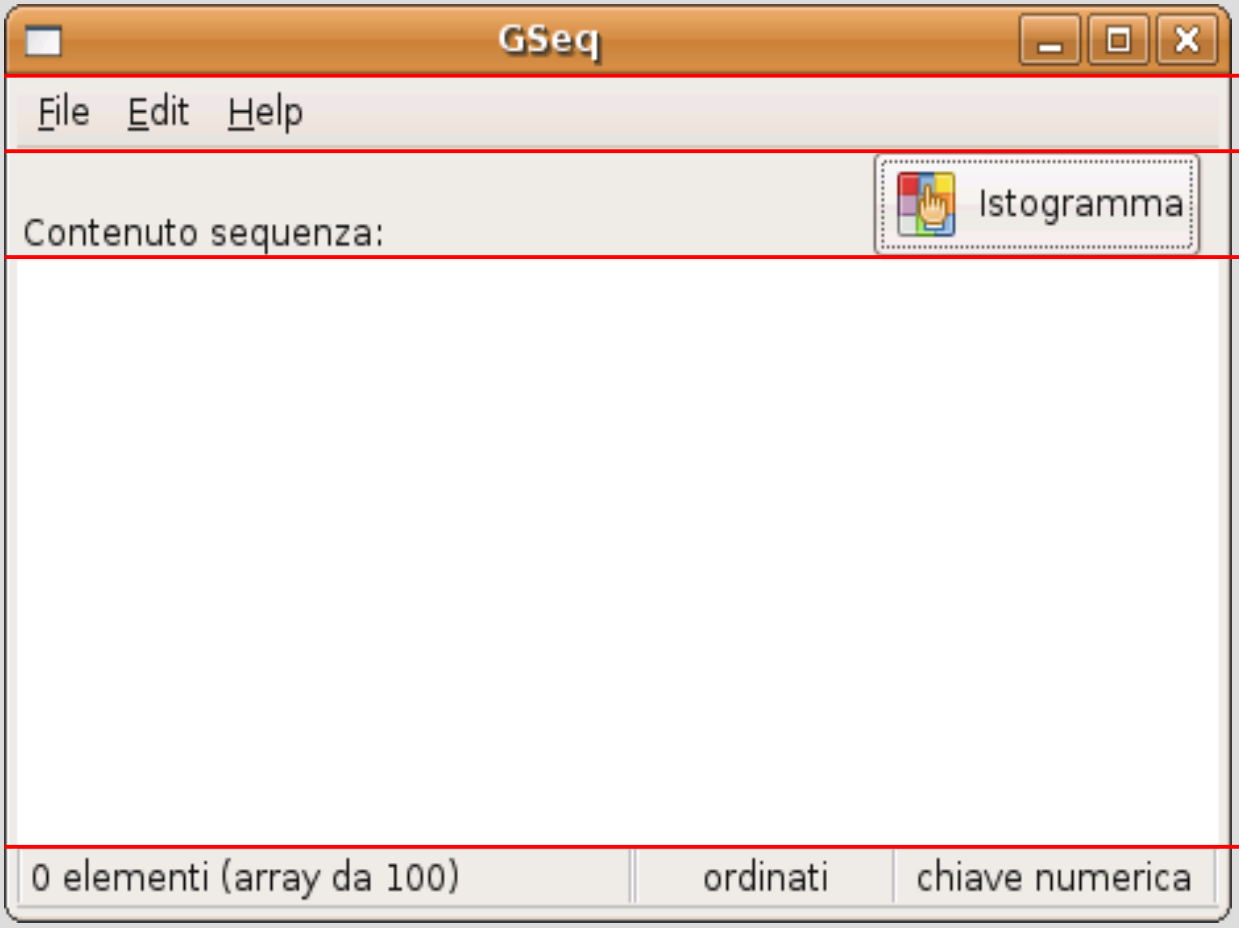
- **Vertical box**

  Sequenza di box consecutivi disposti l'uno dopo l'altro in verticale

Dimensioni scatole

- Da cosa dipenderà poi la dimensione dei singoli box?
- Dalla dimensione del **widget** che metteremo nel box
  - … e da altri parametri che spiegheremo a breve
- Prima di introdurre altri dettagli, proviamo ad inserire i 4 **widget** della finestra principale mediante **Glade**

Finestra principale



Widget

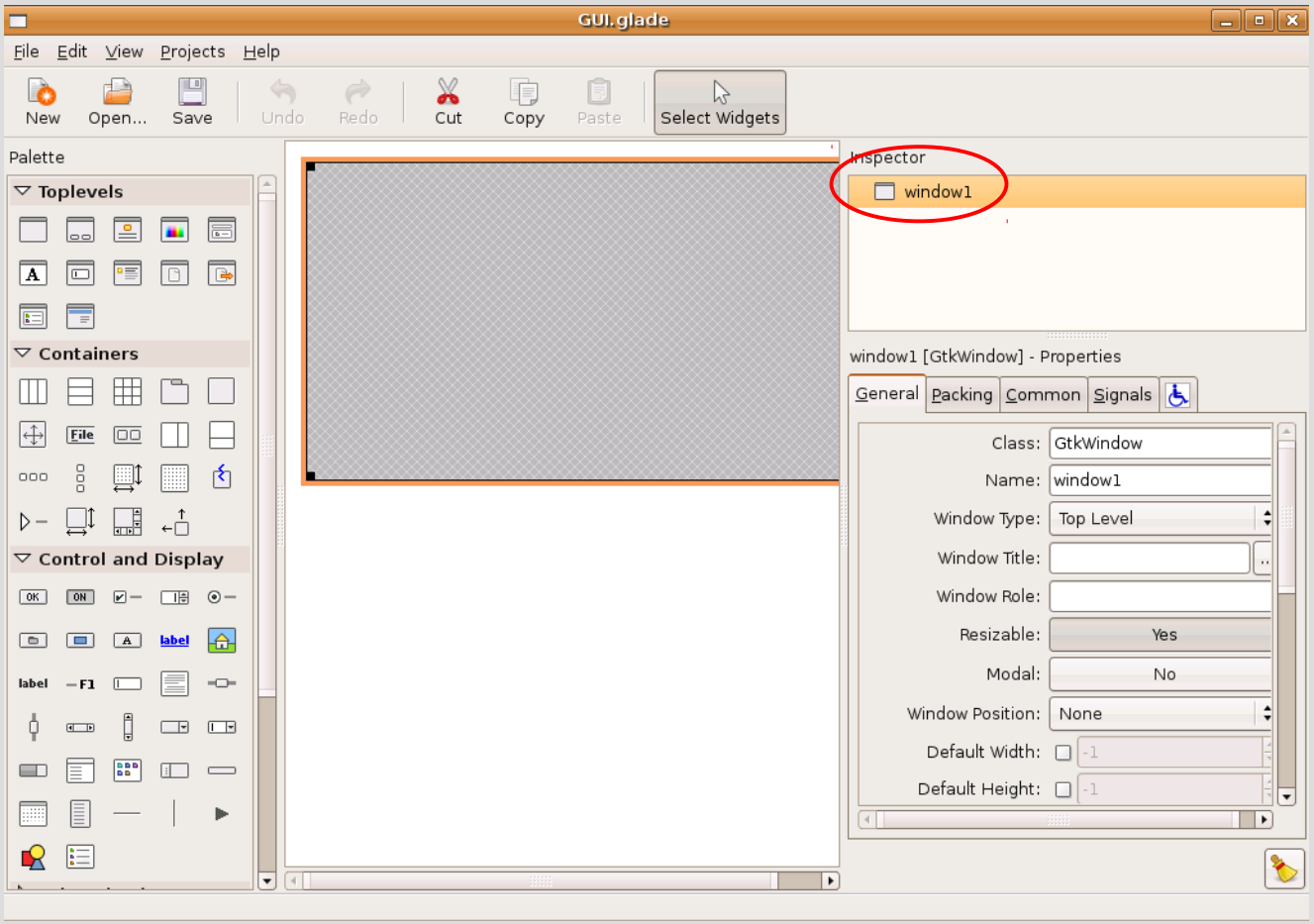
Tali **widget** sono:

- **Barra dei menù (menubar)**
- **Etichetta (label)**
  - Per ora facciamo finta che non vi sia il bottone a lato *Istogramma*
- **Area di testo (textview)**
- **Barra di stato (label)**
  - Per ora supponiamo che sia costituita da una sola etichetta senza separatori

Ci serve un vertical box con 4 box contigui

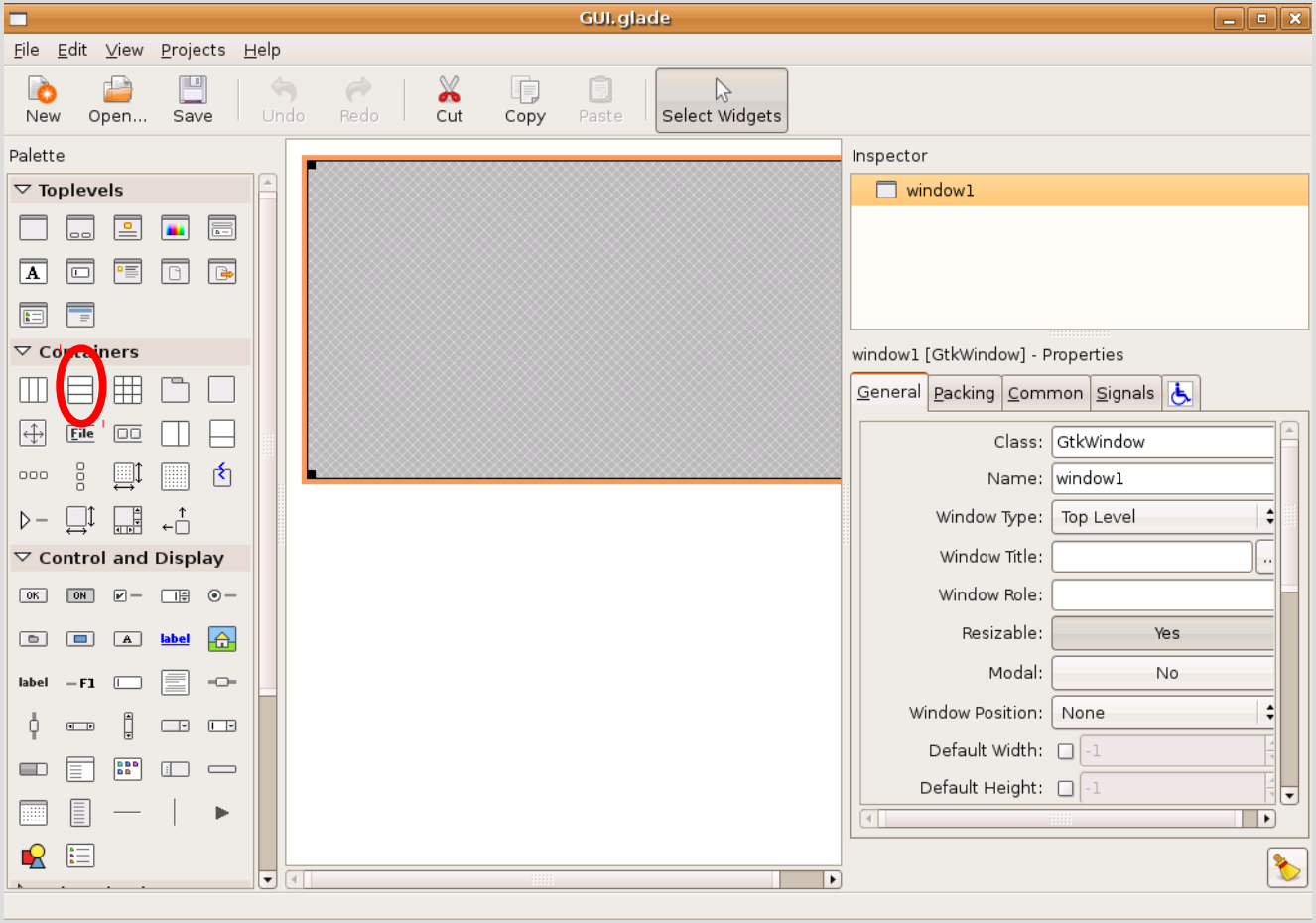
Inizio

- Doppio click sul widget se la finestra a destra non appare



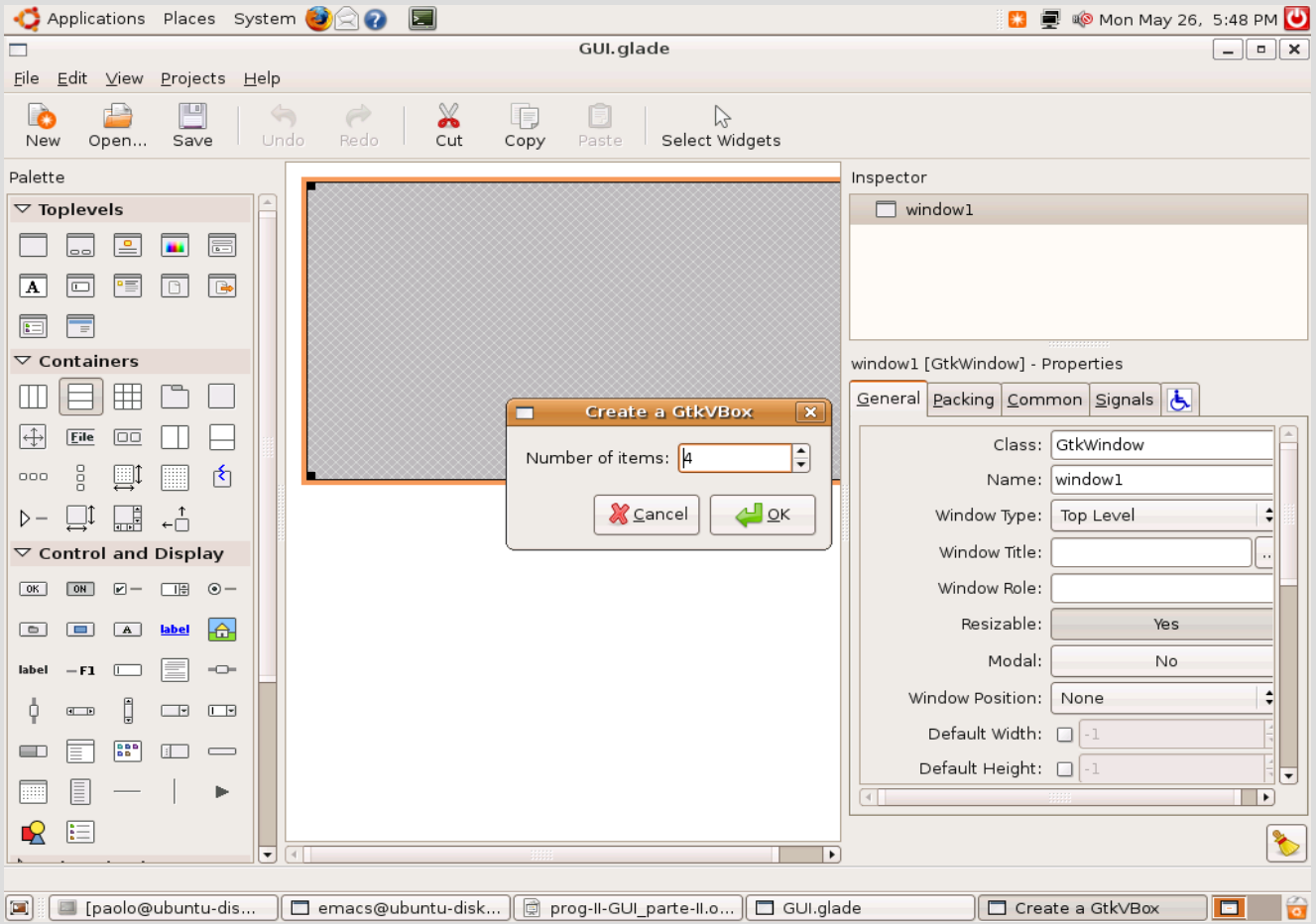
Vertical box

- Selezioniamo vertical box tra i containers



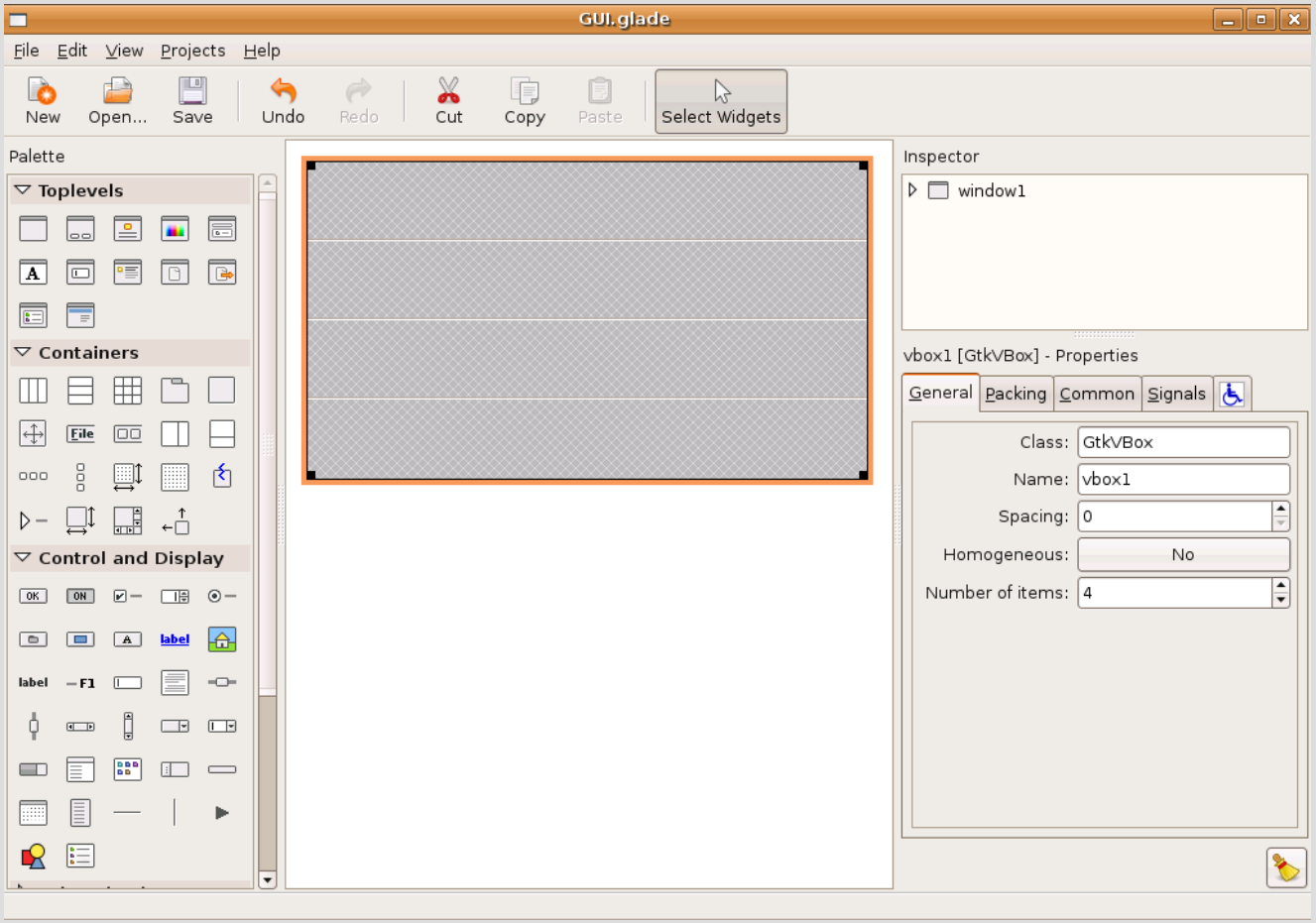
Vertical box

- Clicchiamo nell'area della finestra e scegliamo il numero di box = 4



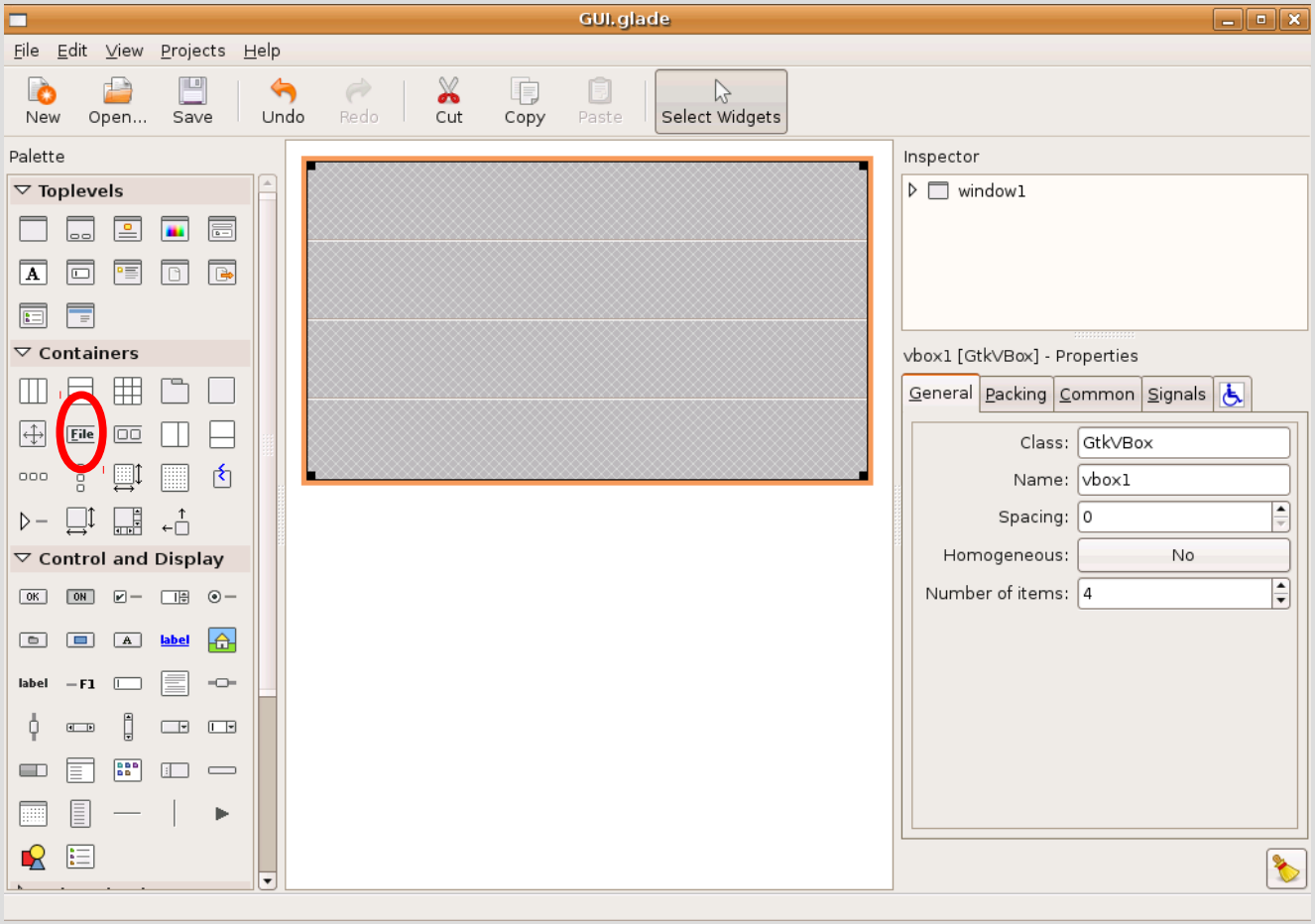
Vertical box

- Vediamo 4 box vuote da riempire



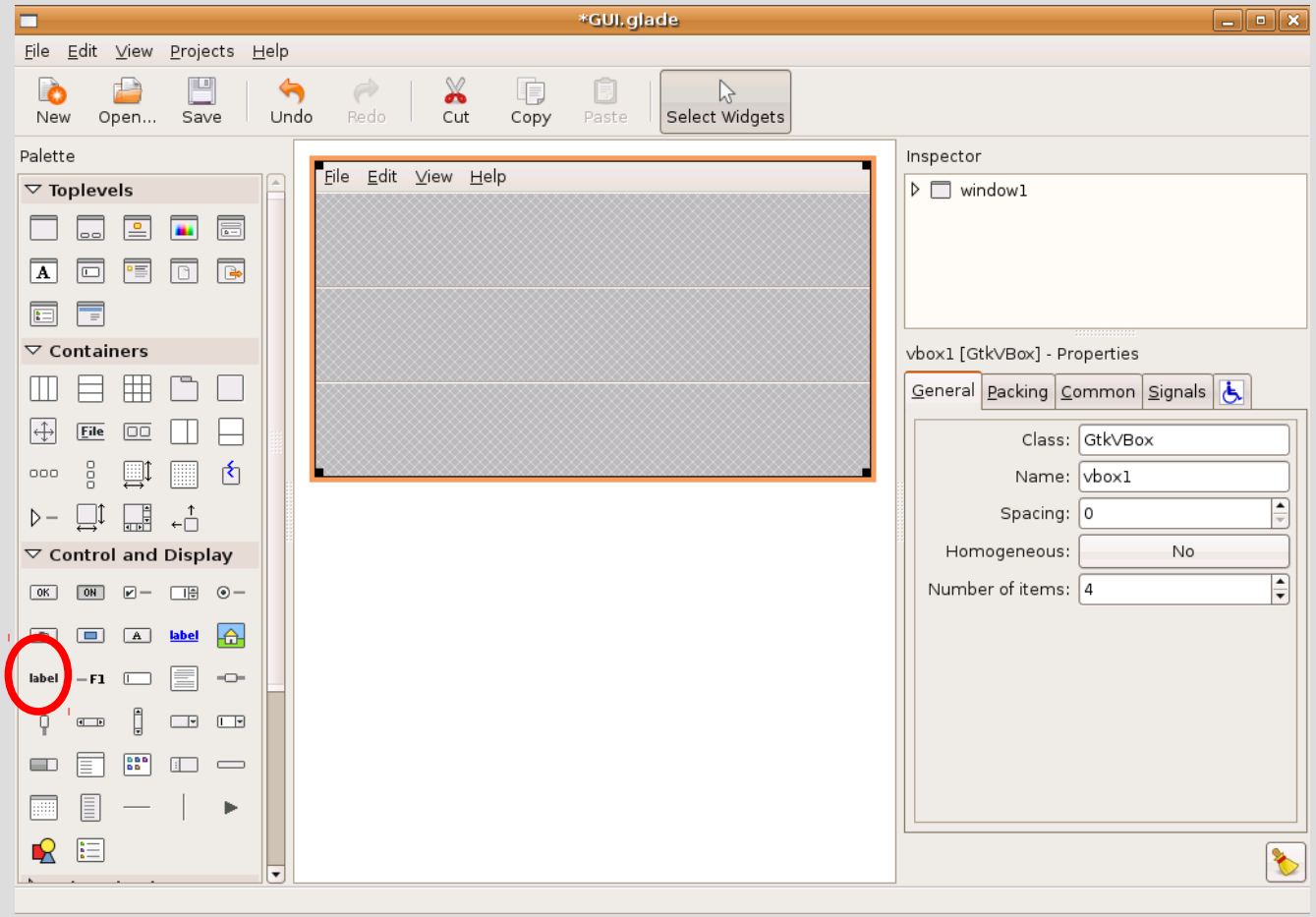
Barra dei menù

- Selezioniamo il **widget menubar** ed inseriamolo nel primo box



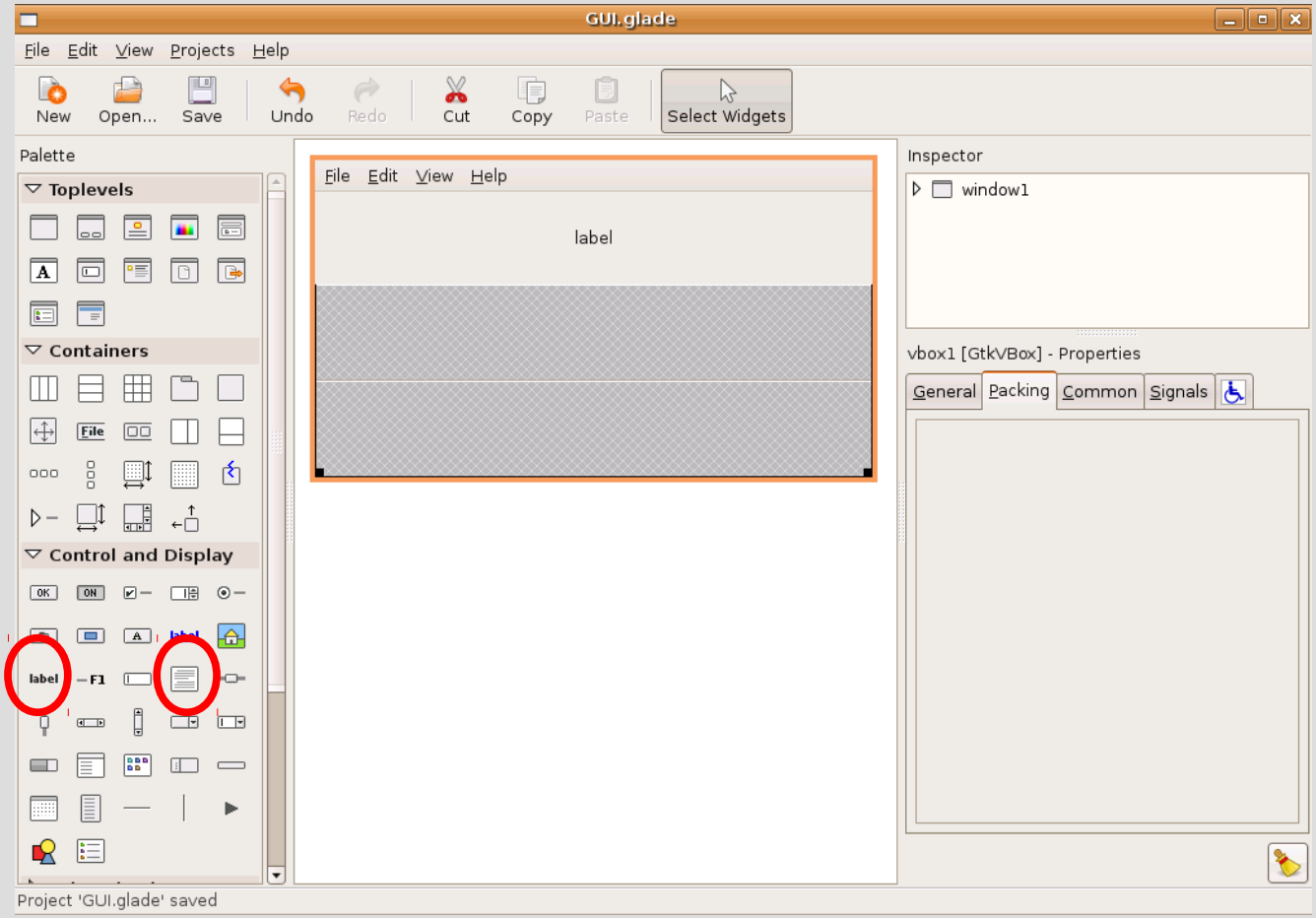
Label

- Selezioniamo il **widget label** ed inseriamolo nel secondo box

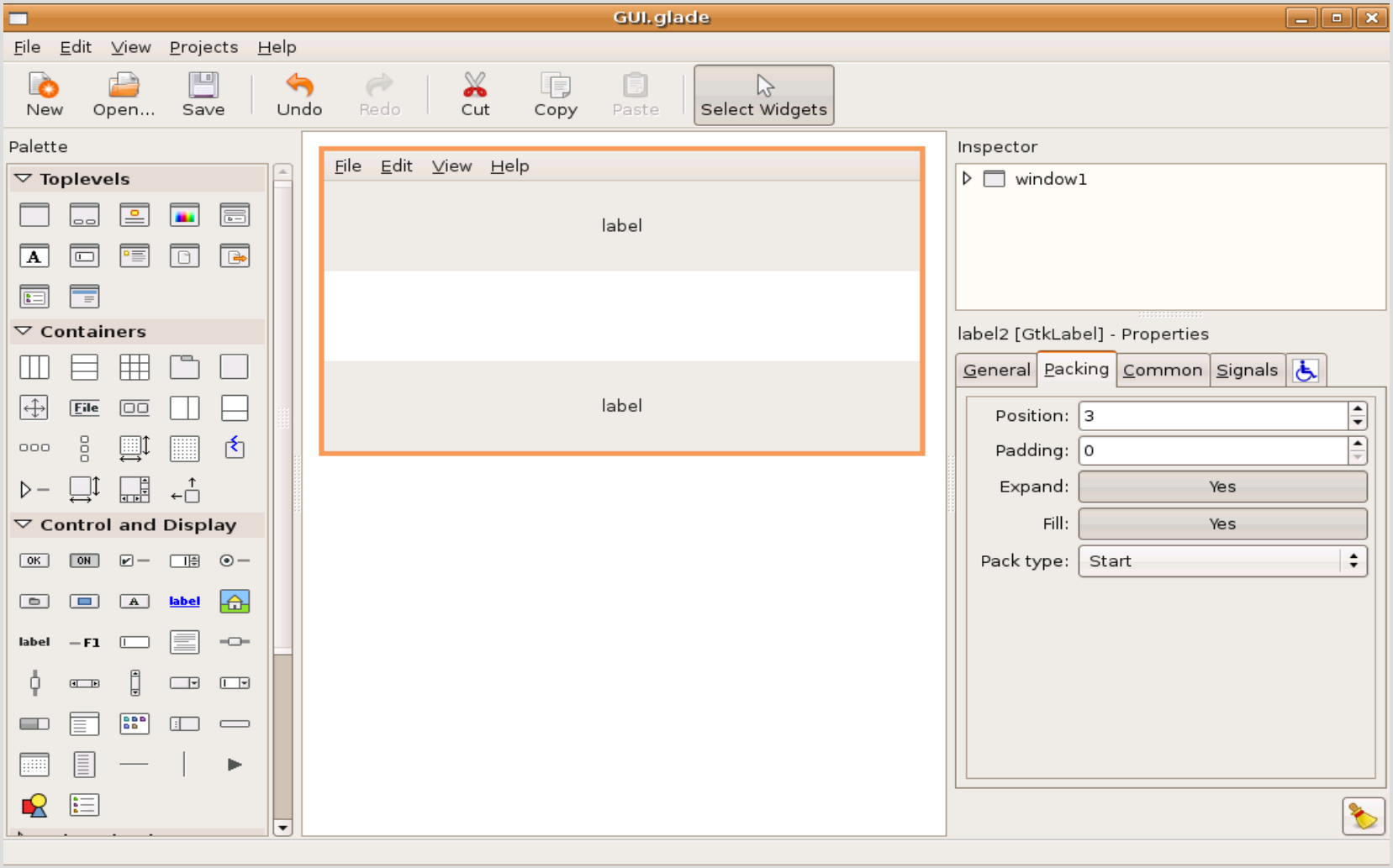


Text view e label

- Selezioniamo i **widget text view** e **label** ed inseriamoli nei rimanenti box



Risultato finale



Inizio modifica main

- Come possiamo vedere all'opera la nuova interfaccia?
- Basta includere i necessari **header file**, ed inserire nel **main** le semplici istruzioni di inizializzazione del **gtk+, caricamento dell'interfaccia ed avvio del main loop** viste nel nostro primo programma di esempio
- Copiamo anche l'**handler** del segnale **delete_event** ed **agganciamolo** mediante **Glade**
- Ricompiliamo ...

Modifica Makefile

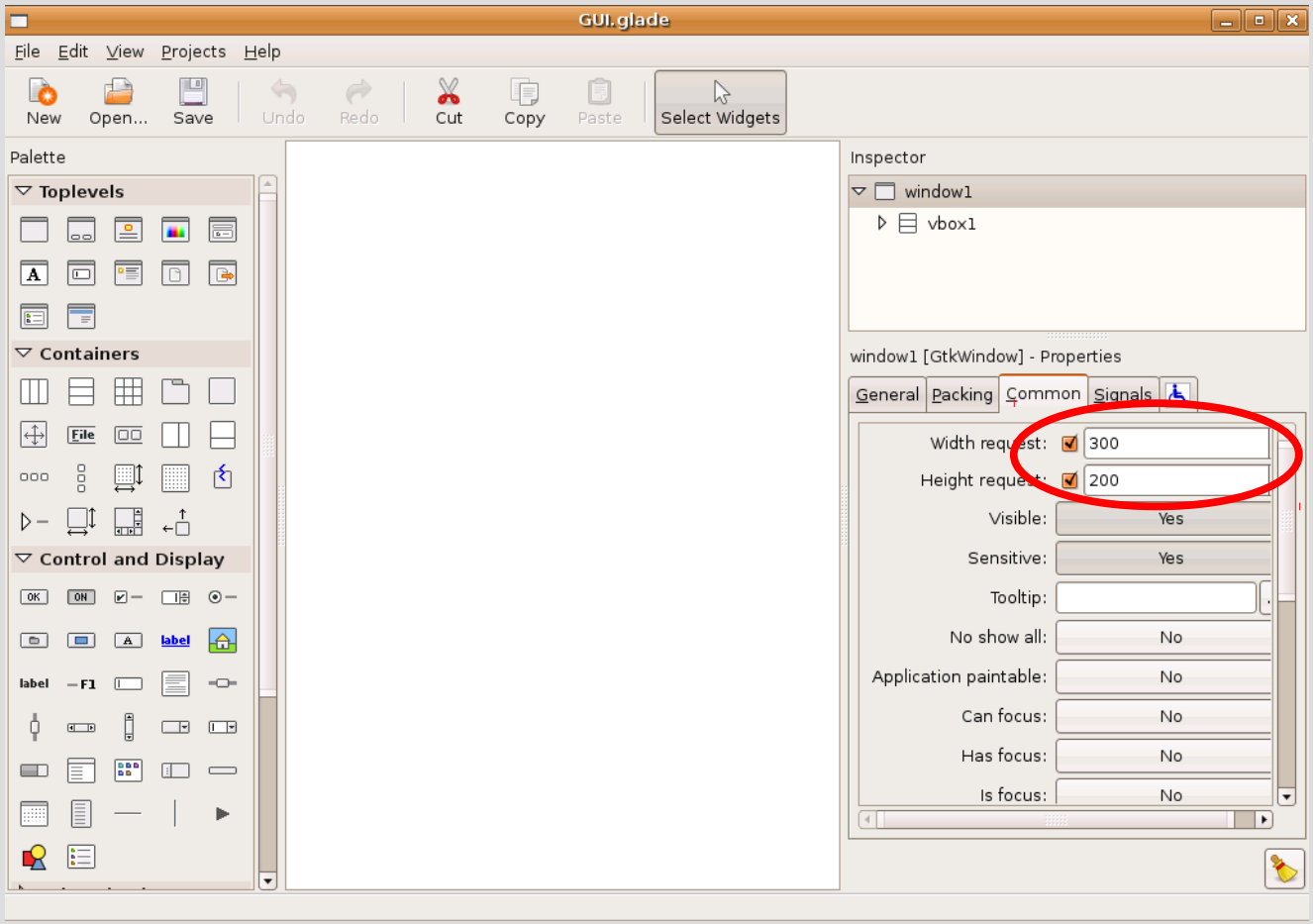
- Probabilmente otterremo molti errori
- Il motivo è che il compilatore non sa dove cercare gli **header file** che servono!
  - Mancano le opportune opzioni **`-I`** (i maiuscola)
  - Mancano anche le opzioni **`-l`** (elle minuscola) per il **linker**
- In breve il **Makefile** va modificato come quello in *progetto_GUI/Makefile*

Dimensione finestra

- Fatta la modifica, ricompiliamo ed eseguiamo
- La finestra che ci appare potrebbe essere molto piccola
- In generale, possiamo scegliere le dimensioni iniziali di un **widget** settando le proprietà **Width Request** ed **Height Request**
- Scegliamo quindi le dimensioni iniziali della finestra come illustrato nella seguente slide

Scelta dimensioni iniziali

- Spuntare le caselle ed immettere i valori come illustrato (***widget window1***)



Dimensione scatole

- Rieseguiamo …
- **Altra nota**, le due label occupano molto più spazio della singola riga che avremmo voluto
- Il vertical box di default cerca di dare ai box in esso contenuti la **stessa dimensione**
- Questa regola può essere forzata mediante la proprietà booleane **expand** che vedremo di seguito (*proprietà dei **widget** che sono contenuti in box – **sezione Packing***)

Proprietà Expand

Expand:

- se **vera**, il box può avere dimensioni maggiori di quelle del **widget** in esso contenuto (legate ad altre eventuali regole) e arrivare a occupare tutto lo spazio assegnato
- se **falsa**, il box è vincolato ad avere esattamente le dimensioni del **widget**

Proprietà Fill

Fill

- Dipende direttamente dal valore di **expand**: considerato solo se **expand** è settato a vero
  - se **vera**, le dimensioni del **widget** sono sempre uguali a quelle del box in cui è contenuto
  - se **falsa**, il **widget** conserva le proprie dimensioni (legate ad altre eventuali regole) anche se il box è più grande
    - in questo caso rimane dello spazio vuoto nel box

Verifica

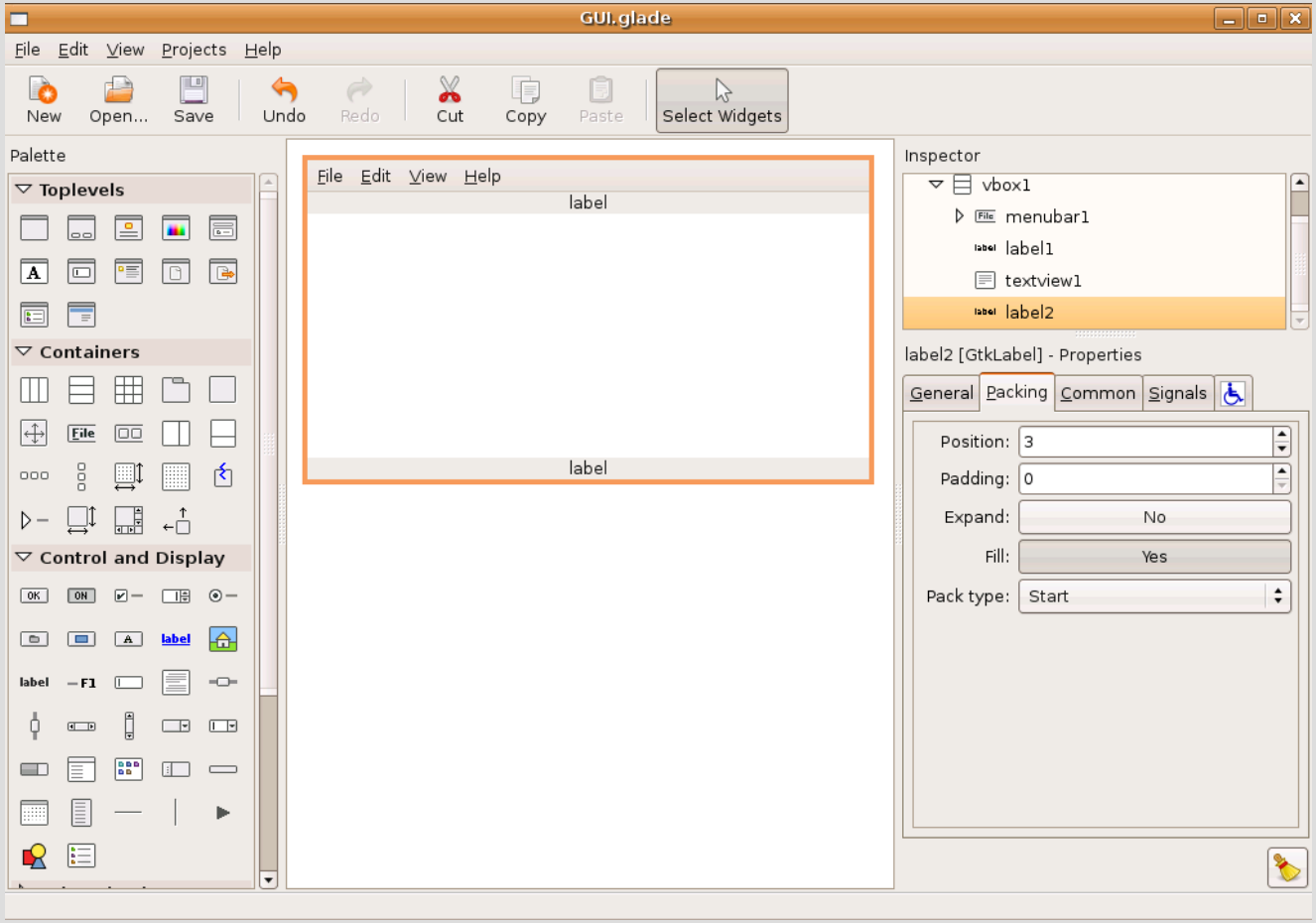
- Quindi allo stato attuale nel nostro esempio dovremmo trovare per le **due label** e per il **widget menubar** la proprietà **expand** settata a **vero** (Yes nella nomenclatura di Glade)
- Anche la proprietà **Fill** dovrebbe essere vera
- *In Glade, per ciascun **widget** troviamo tale proprietà nella scheda **packing***
- **Provate a settare Fill nelle label a FALSO**
  - *Vedete che la label occupa meno della box assegnata e resta spazio vuoto*

Modifica

- Mettiamo a falso il valore di **expand** per entrambe le **label** e ricontrolliamo
- Note:
  - Si può selezionare un **widget** o cliccando sopra la sua immagine nell'area centrale o selezionandolo nella finestra **Inspector**
  - Se un **widget window** presente nell'area **Inspector** non appare nell'area centrale, per farlo apparire basta fare doppio click sul suo nome nell'area **Inspector**

Risultato finale

- Se tutto è andato bene dovremmo avere:



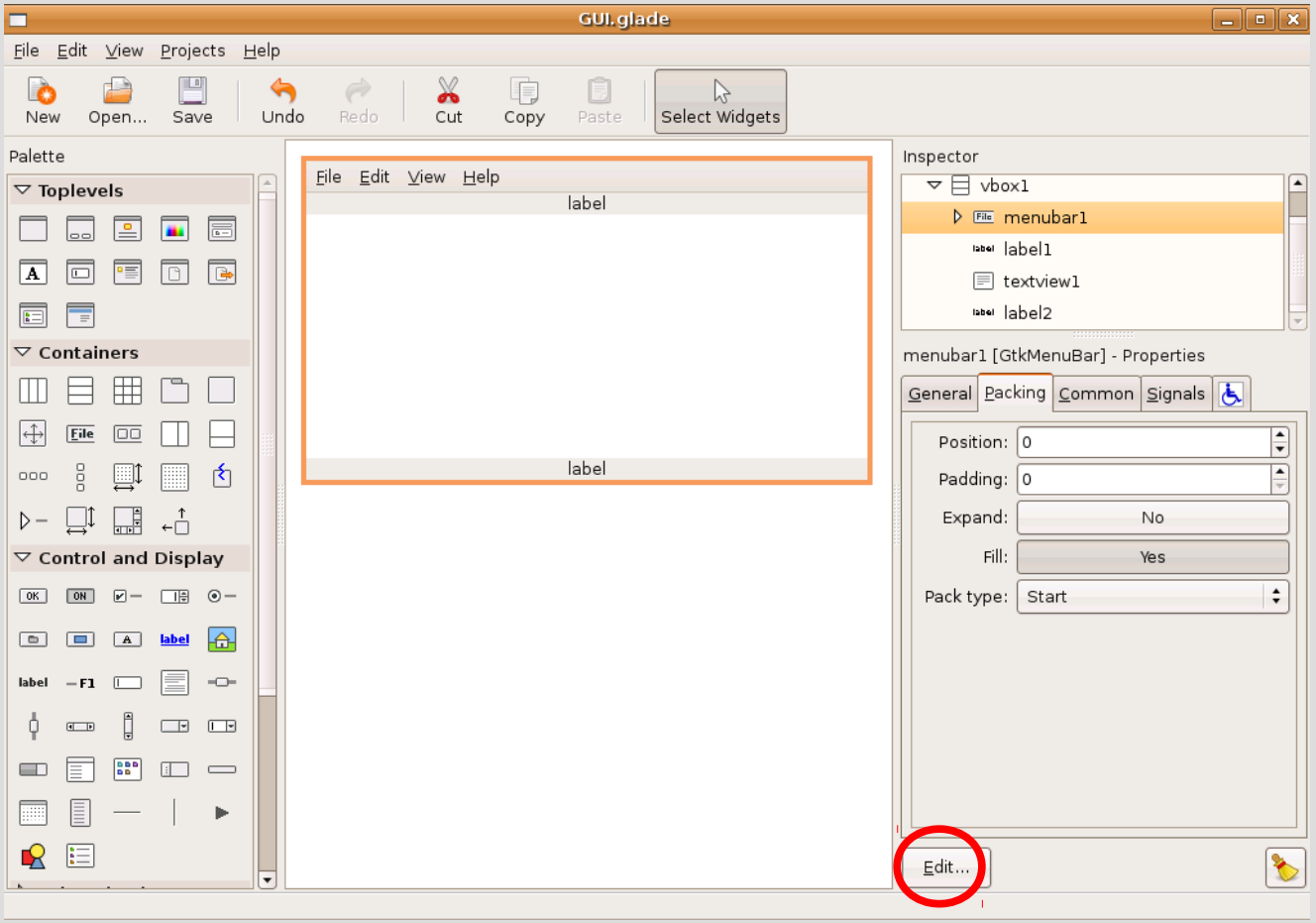
Menubar (1)

- Passiamo ora alla **barra dei menù**
- Ci sono voci inserite di default nel menubar
- Vogliamo aggiungere/rimuovere voci/sottovoci nella nostra interfaccia (Es. View, Edit)
- Vediamo come sistemare il **menubar**

Nota: le nuove versioni di Unity in Ubuntu utilizzano una menubar di sistema nell'estremita' superiore del desktop. Per tornare alla visualizzazione classica, con menubar in cima ad ogni finestra, occorre disinstallare i seguenti pacchetti (ad es. con Synaptic Package Manager): appmenu-gtk e appmenu-gtk3

Menubar (2)

Selezioniamo menubar e premiamo il bottone **Edit** *(oppure click col destro su menubar e poi Edit)*

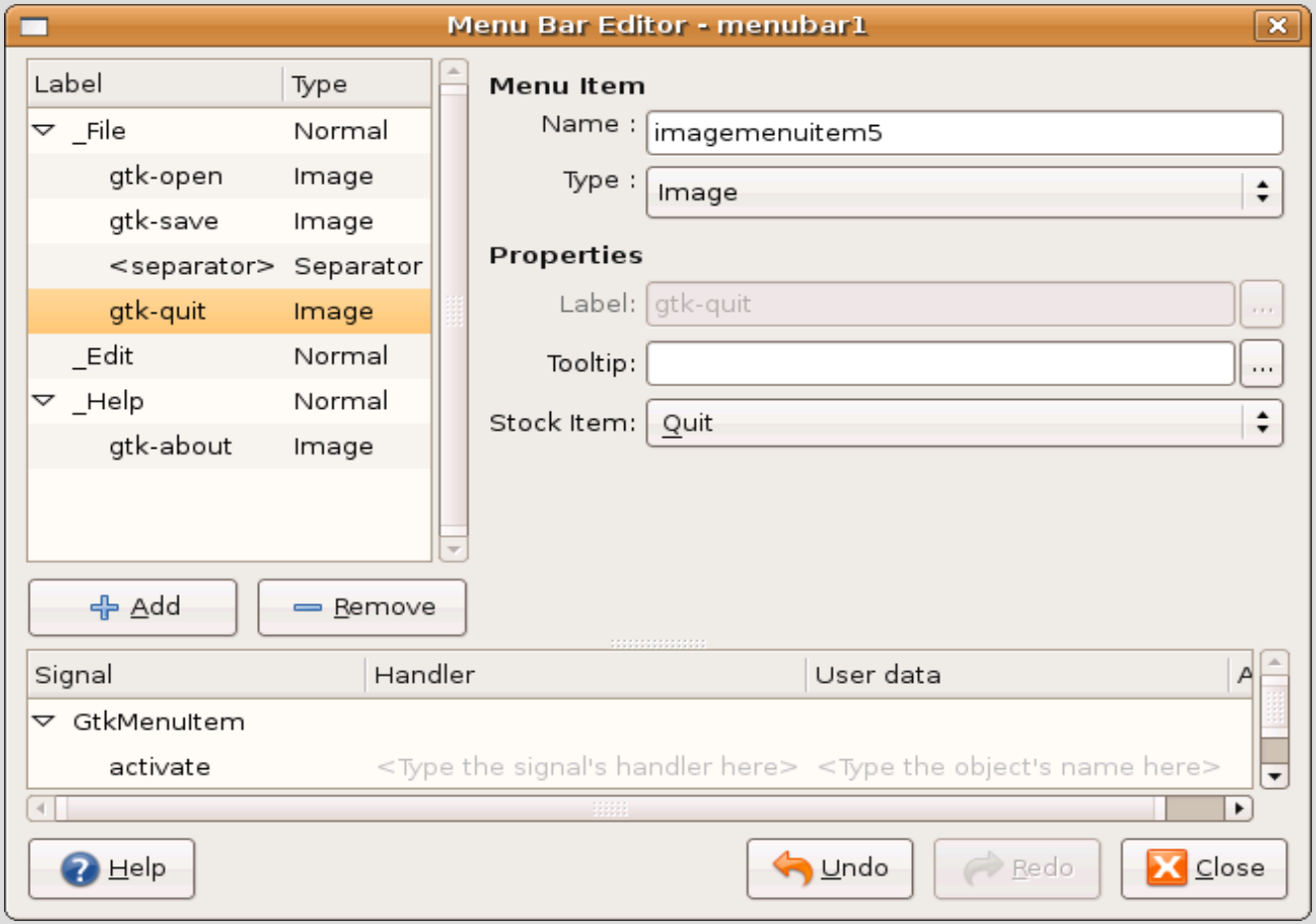


Eliminazione elementi (1)

- Clicchiamo sulla scheda **Hierarchy**
- Nella finestra che ci si apre **eliminiamo** innanzitutto tutte le voci che non vogliamo compaiano nella nostra interfaccia
  - Es. View e le sottovoci di Edit (Cut, Copy, Paste)
- Per eliminare una voce basta selezionarla e premere il tasto **Cancella** sulla tastiera oppure usare il pulsante **Remove**

Eliminazione elementi (2)

- Dovremmo avere questa situazione



Aggiunta voci

Aggiungiamo quindi le voci che mancano

- Per aggiungere un elemento basta per esempio cliccare col pulsante destro del mouse sull'elemento che lo dovrà precedere e scegliere:
  - **Add Item** se si vuole inserire un elemento allo stesso livello di quello precedente
  - **Add Child Item** se si vuole inserire un elemento ad un livello inferiore (un suo figlio quindi)

Aggiunta separatori

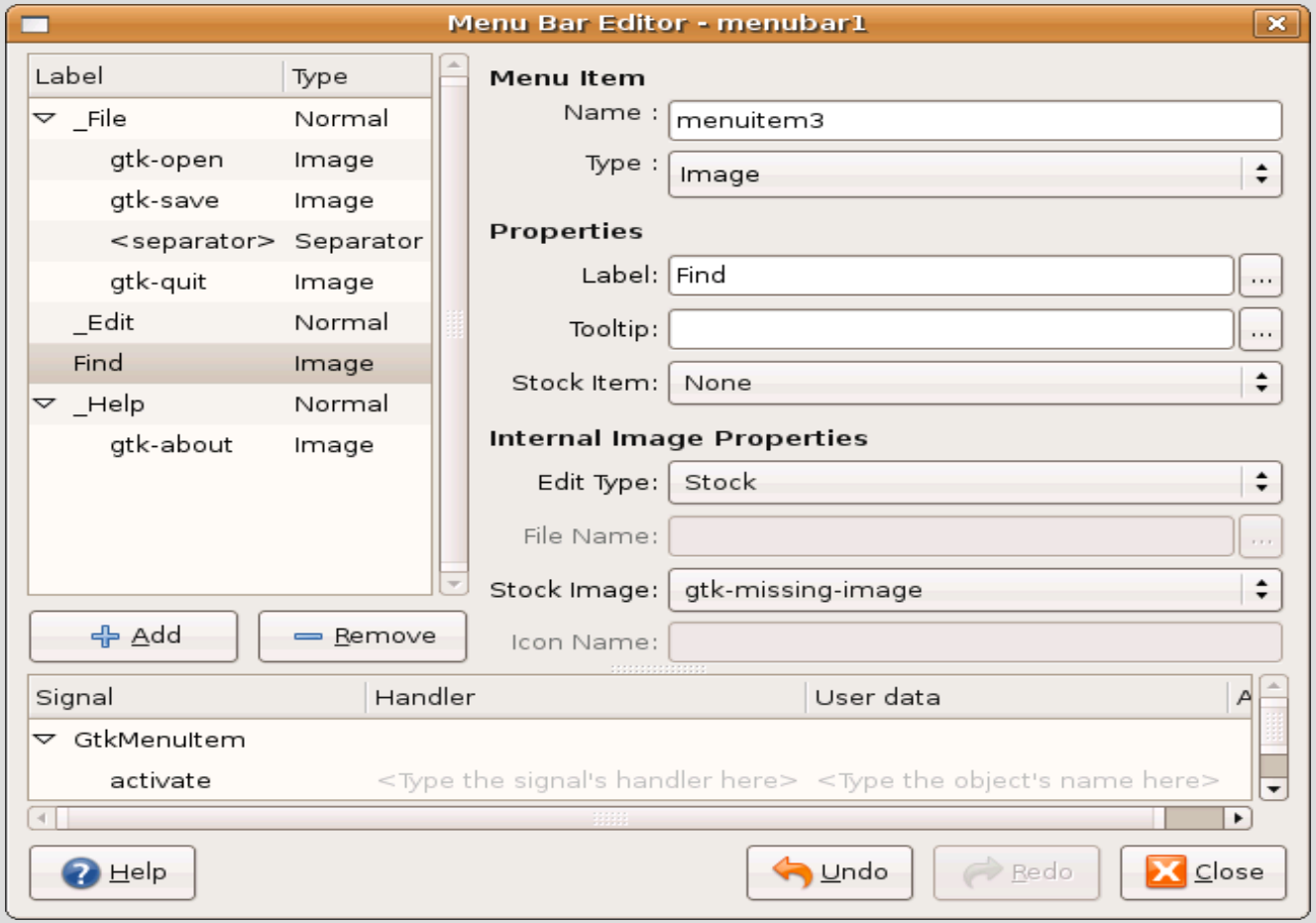
- Per aggiungere separatori (linee) tra le voci del menù cliccare col tasto destro sull'elemento che deve precedere il separatore e scegliere la voce **Add Separator**

Proprietà voce del menubar

- Nel campo **Tooltip** si può eventualmente inserire il messaggino che apparirà se ci si ferma con il puntatore sul **widget**
- Una proprietà importante è **Type**
  - Selezioniamo **Image** per la voce **Find** per ottenere una voce di menù con immagine accanto
  - A questo punto possiamo scegliere la voce di menù tra quelle preconfezionate in **Stock Item**

Situazione corrente

- Dovremmo avere qualcosa di simile:



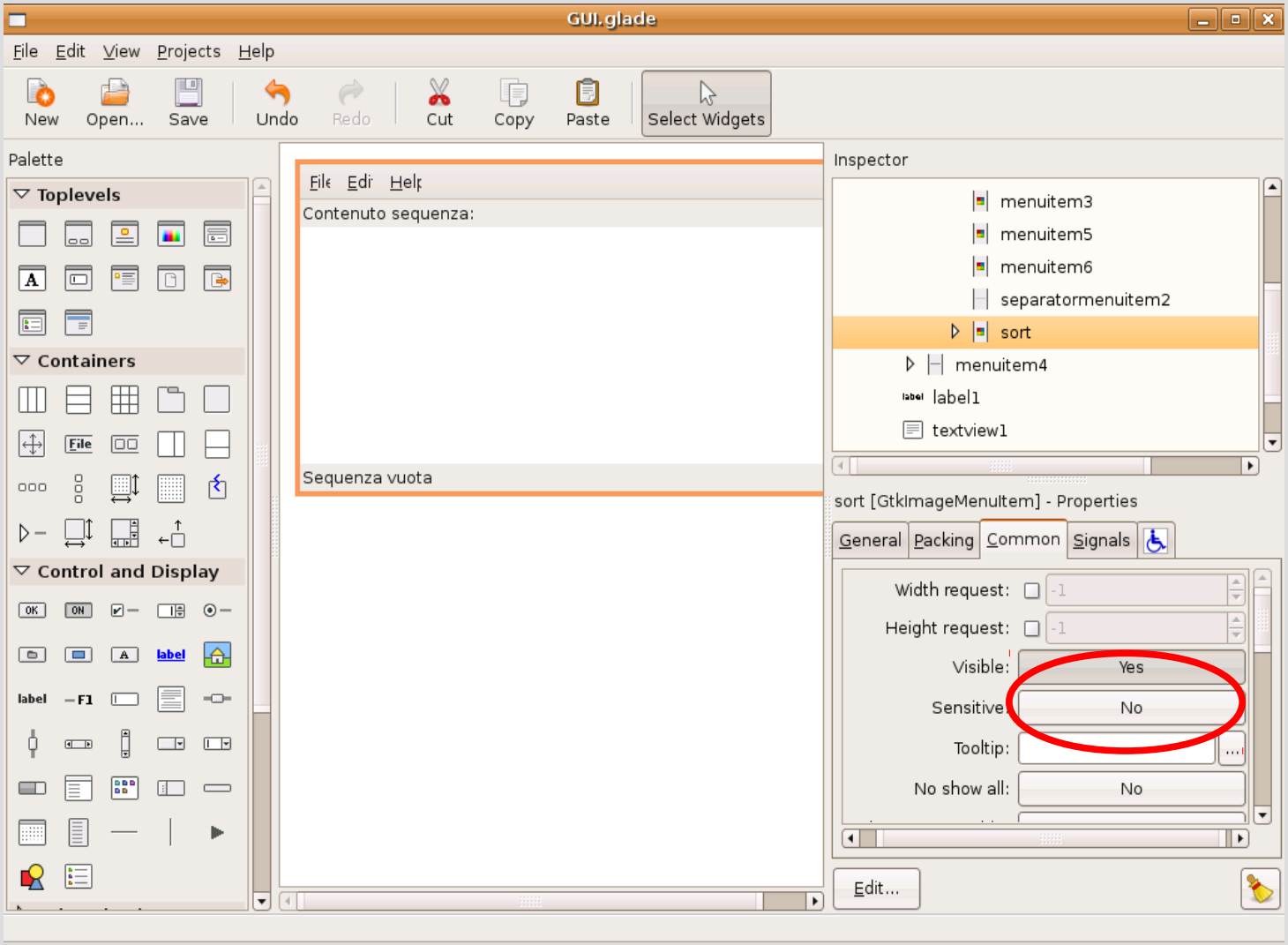
Voci pre-confezionate

- Possiamo scegliere di usare voci pre-confezionate per le voci del menù **Edit**
- Per farlo basta selezionare la voce desiderata dal bottone a scelta multipla di fianco all'etichetta **Stock Item**
  - Ovviamente dopo aver scelto **Image** per il **Type**
- Settare a **YES** la voce **Always show image** se presente

Voci inattive

- Per rendere **inattiva una voce** basta mettere a **No la proprietà Sensitive** del **widget** che rappresenta quella voce del menù
  - La proprietà **Sensitive** si trova nella scheda **Common**
  - La voce del menù risulta non cliccabile
  - Mostrato nella prossima slide per la voce **Sort**

Voce Sort disattivata



Label

Passiamo ora alla **label** sottostante la **barra dei menù**

- Immettiamone il contenuto “Contenuto sequenza:”
- Vogliamo allinearne il testo a sinistra
- Per posizionare la label (ed in generale ogni **widget**) all'interno del **widget contenitore** bisogna padroneggiare le **4 proprietà** che andremo a vedere
  - Proprietà contenute nella scheda **General**

Allineamento di un widget

X align

- Allineamento orizzontale all'interno del **widget** contenitore
- Valori continui da 0 a 1: 0 -> allineato a sinistra, 0.5 centrato, 1 -> allineato a destra

Y align

- Identico ad **X align** ma in senso verticale

Padding

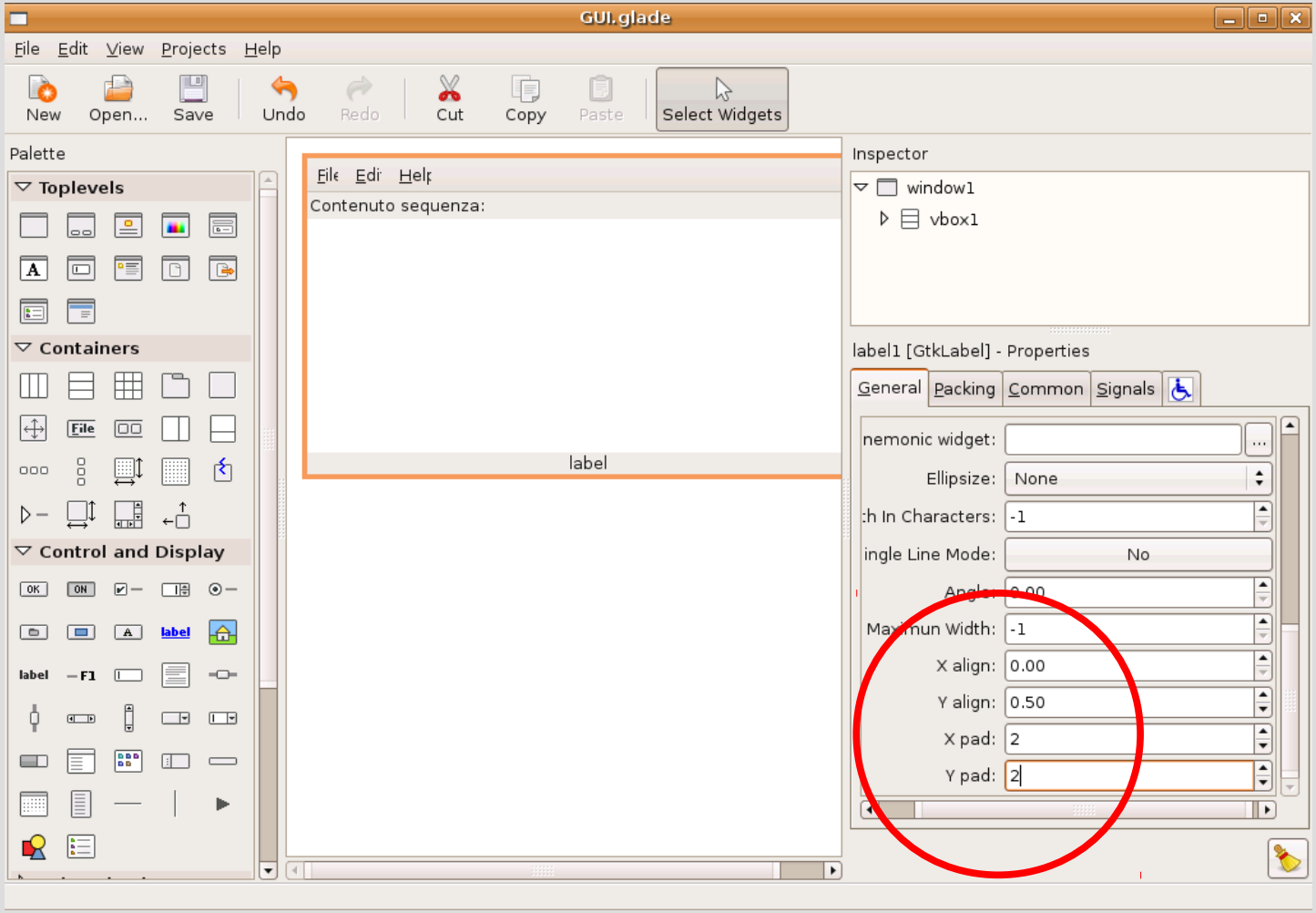
X pad

- Spazio vuoto in pixel da aggiungere a destra ed a sinistra del **widget** all'interno del **widget contenitore**

Y pad

- Spazio vuoto in pixel da aggiungere sotto e sopra del **widget** all'interno del **widget contenitore**

Esempio


GUI.glade
File Edit View Projects Help
New Open... Save Undo Redo Cut Copy Paste Select Widgets
Palette
Toplevels
Containers
Control and Display
File Edit Help
Contenuto sequenza:
label
Inspector
window1
vbox1
label1 [GtkLabel] - Properties
General Packing Common Signals
nemonic widget:
Ellipsize: None
ch In Characters: -1
ingle Line Mode: No
Maximun Width: -1
X align: 0.00
Y align: 0.50
X pad: 2
Y pad: 2

Handler voce del menù (1)

- Creiamo ora un **handler** per la voce di menù **Add**
- Quando viene selezionata una voce di menù (**GtkMenuItem**) emette il segnale **activate**
  - Cercare sul manuale il prototipo che deve avere il relativo **handler**

    /usr/share/doc/libgtk2.0-doc/gtk/index.html
    - Seguire il link **III. GTK+ Widgets and Objects**
    - Cercare il link al **widget GtkMenuItem**
    - Cercare in sezione **Signal Details** il segnale **activate** e il prototipo del relativo **handler**

Handler voce del menù (2)

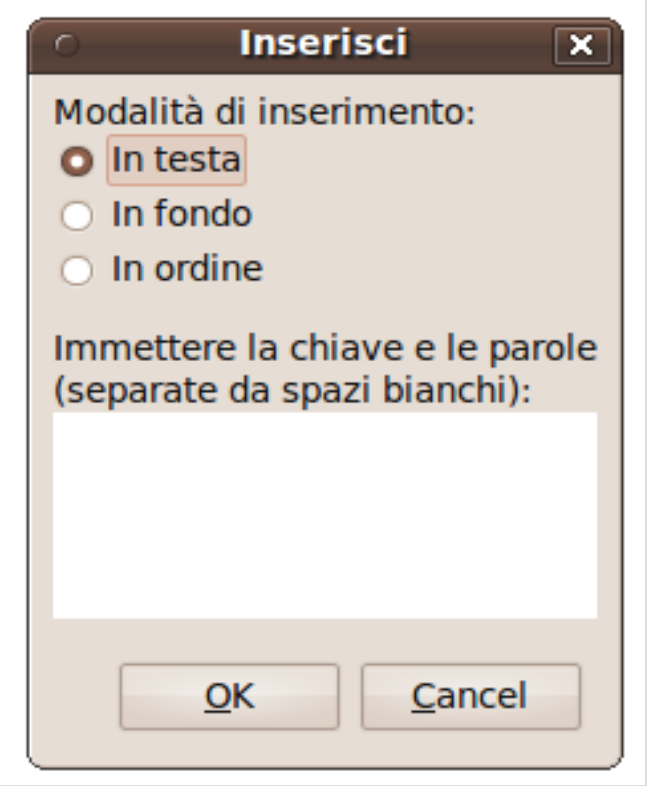
- Inserire nel codice un **handler** che rispetti tale prototipo e stampi solo un messaggio su **stdout**

```
extern "C"  void handler_activate_Add (GtkMenuItem
                                    *menuitem, gpointer user_data)
{ cout << “Handler Add “ << endl;
}
```

- Agganciare **l'handler** mediante **Glade**
- Verificare il funzionamento

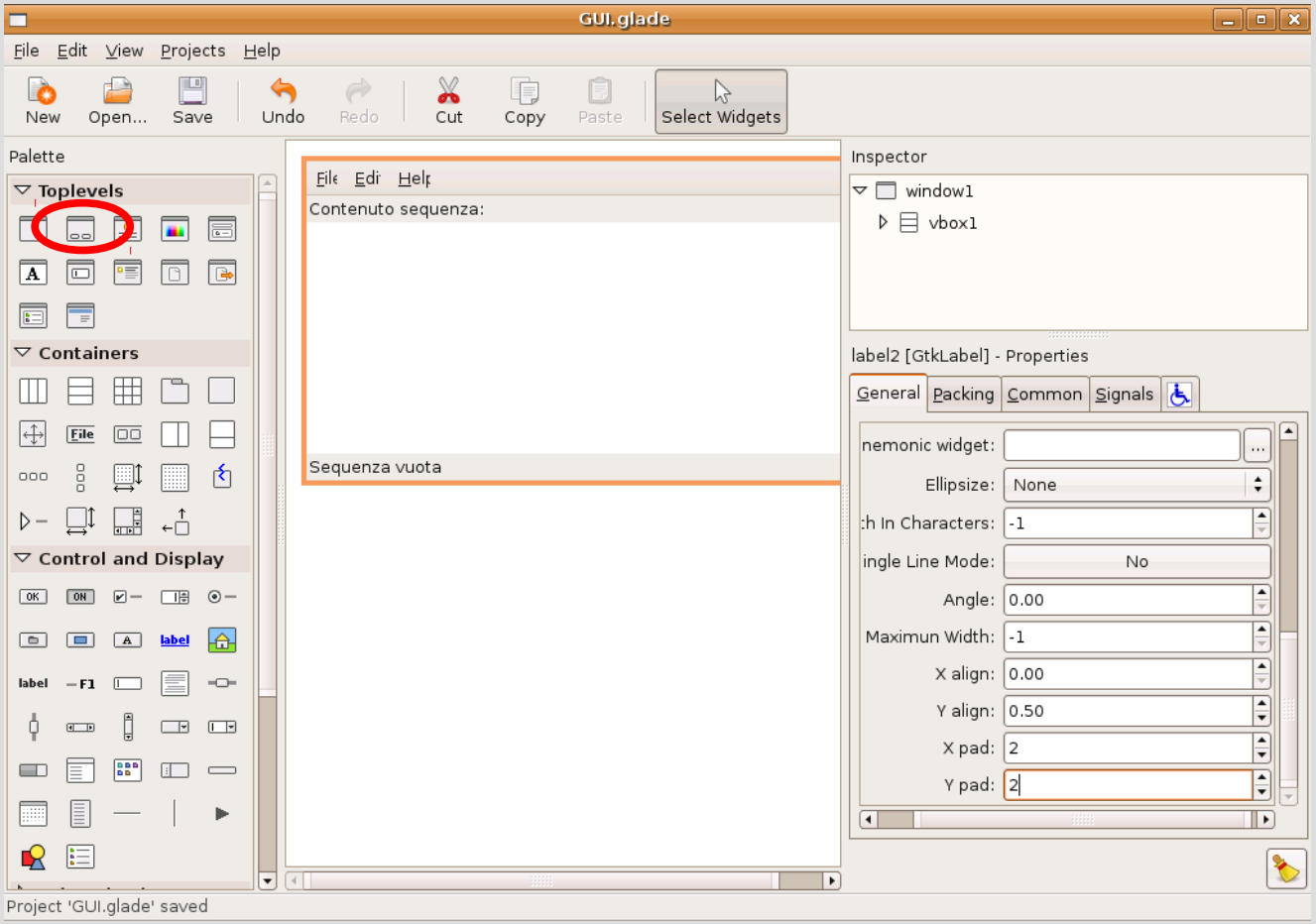
Finestra di dialogo

- In realtà la voce di menù **Add** deve far aprire una finestra in cui scegliere la modalità di inserimento ed inserire le parole di cui è costituito un elemento
- Una finestra del genere è chiamata **finestra di dialogo**
- Cominciamo dal disegnare tale finestra con **Glade**
  - Il **widget** corrispondente è chiamato **Dialog Box**
- Poi vedremo come farla apparire!



Aggiungere finestra di dialogo

- Per aggiungere una finestra di dialogo clicchiamo su corrispondente **widget**



Riempimento finestra

Comparirà la finestra di dialogo nell'area centrale

- **NOTA**: ha la proprietà **visible** a **false**, per non comparire subito
- Ha già predisposti due spazi vuoti:
  - Per la parte in cui immettere le informazioni
    - Qui inseriamo le **label**, i **radio button** e la **textview** come da interfaccia di riferimento
  - Per i bottoni
    - Qui inseriamo i bottoni **OK** e **Cancel**

Collocazione elementi

- Per riempire la parte superiore della finestra, progettare prima la collocazione in termini di **box** e inserire i **box** opportuni
  - Servirà un vertical box
  - *Nota: ogni radio button occupa un box*
- Poi inserire **label, radio button e text view** nei **box** opportuni
  - Settiamo le dimensioni e mettiamo le scritte giuste nelle label

Radio Button (1)

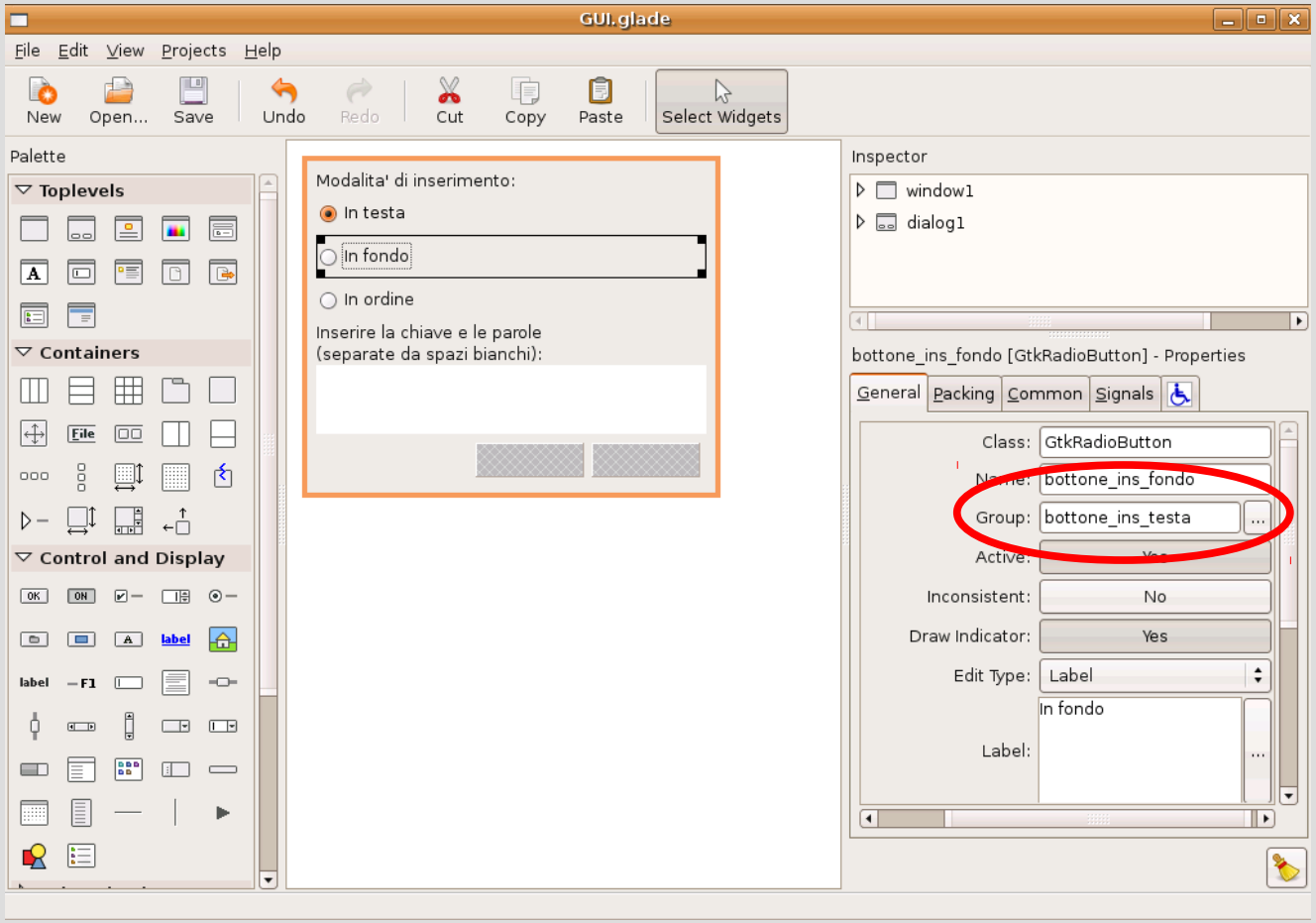
- Per far sì che un gruppo di **radio button** funzioni opportunamente (*sia possibile selezionarne solo uno alla volta*) bisogna comunicare a **GTK+** che si tratta di un **gruppo**
- Per ottenere questo obiettivo tramite **Glade:**
  - si sceglie uno dei bottoni come rappresentativo del gruppo
  - si fa a puntare a tale bottone il campo **Group (scheda General)** in ciascuno degli altri bottoni
- Facciamo le cose con ordine...

Radio Button (2)

- Diamo un **nome sensato** a ciascun bottone
- Mettiamo la **label** giusta (il testo che apparirà sull'interfaccia) per ciascun bottone
- Facciamo puntare il campo **Group** degli ultimi due bottoni al primo bottone
  - Nel campo **Group** possiamo inserire manualmente il nome del bottone rappresentativo del gruppo, oppure selezionarlo graficamente premendo sul bottone con i puntini di sospensione a lato
- Il risultato è mostrato nelle seguenti slide

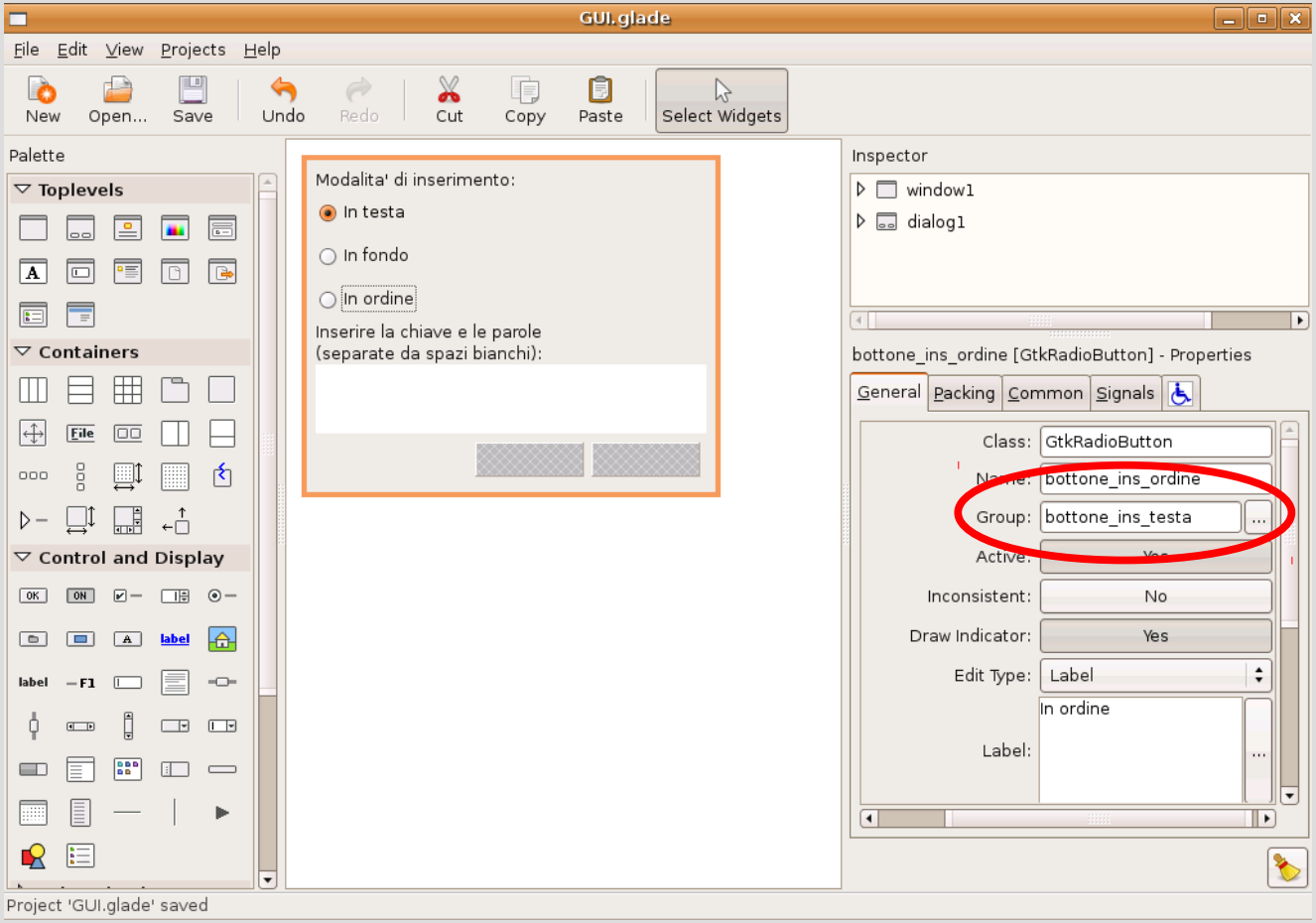
Gruppo radio button (1)

- Il secondo bottone è associato al primo



Gruppo radio button (2)

- Anche il terzo bottone è associato al primo

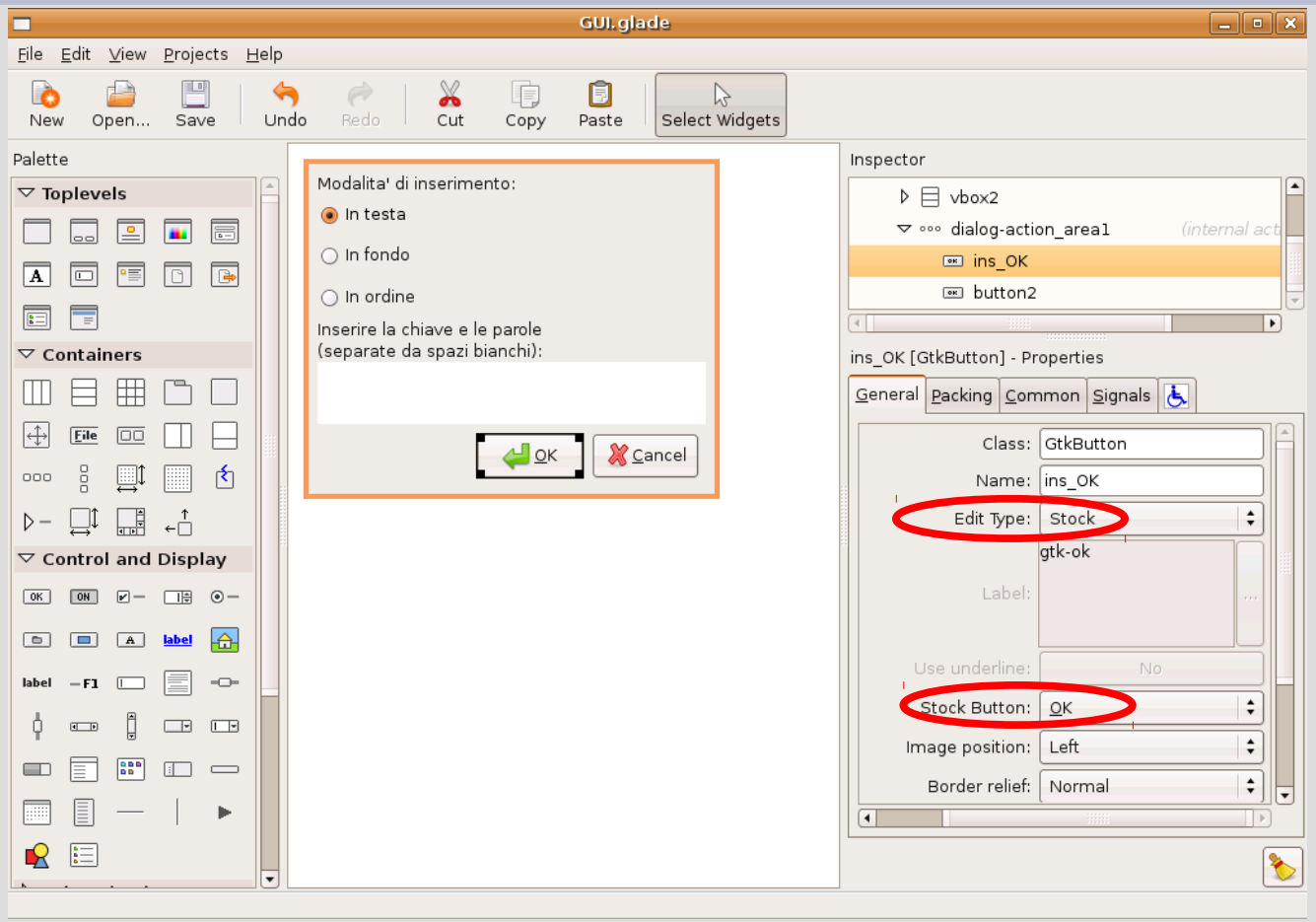


Bottoni semplici

In quanto ai bottoni semplici, inserire il bottone **OK** ed il bottone **Cancel predefiniti**

- Inserire prima i bottoni (widget button)
- Poi per ciascun bottone scegliere **Stock Item** per **Edit Type** e poi scegliere lo **Stock Button** desiderato (Scheda General)
- La configurazione del bottone **OK** è mostrata nella prossima slide

Bottone OK



[Nota: nelle ultime versioni di GTK, i bottoni non hanno immagini associate. Occorre usare box, immagini e label..]

Completamento

- Applicare tutto quello che si è appreso sulle proprietà per formattare per bene tutti i campi della finestra, verificando che si comportino bene quando si modificano le dimensioni della finestra stessa attraverso Glade
- Eventualmente assegnare la dimensione iniziale della finestra

Text view - osservazioni

- Che succede se inseriamo righe molto lunghe nella **text view**?
  - Preferiremmo che andasse a capo automaticamente: basta settare la proprietà **Wrap Mode** (Scheda General) a **Word**
- E se inseriamo molte righe?
  - Preferiremmo che apparissero le barre di scorrimento
  - Per ottenere questo obiettivo basta inserire la **text view** in una **scrolled window** anziché in una semplice box

Inserimento in Scrolled Window

Per ottenere questo obiettivo con **Glade** possiamo:

- Tagliare il **widget text view** (*Cut*)
- Inserire al suo posto una **Scrolled Window (Tipo Container)**
- Inserire una **text view** nella **Scrolled Window**

Far apparire la finestra

- Vogliamo ora fare apparire la **finestra di dialogo** mediante l'**handler** della voce di menù **Add**
- A tale scopo dobbiamo sapere come compiere **due nuove azioni**
  - Accedere al **widget** che rappresenta la finestra di dialogo
  - Far apparire il **widget**

Accesso ai widget (1)

Con **GtkBuilder**, tale operazione si ottiene utilizzando la funzione **gtk_builder_get_object**, a cui si passano come parametri:

- Il puntatore all'oggetto di tipo **GtkBuilder** ritornato dalla funzione **gtk_builder_new** (che abbiamo usato nel **main** per caricare l'interfaccia grafica)
- Il nome con cui abbiamo chiamato in Glade il **widget** a cui si deve accedere

Accesso ai widget (2)

- La funzione **gtk_builder_get_object** ritorna un puntatore al **widget** cercato
- **ATTENZIONE:** come tipo di ritorno ha un puntatore a **oggetto generico**, ossia **Gobject***
- Se dobbiamo usare il **widget** come **oggetto generico** possiamo quindi usare:

```
GObject *widget_da_accedere =
gtk_builder_get_object(builder,"nome_widget");
```

Widget specifico

- Se invece ci serve il tipo esatto del **widget**, possiamo fare così:

```
TipoWidgetEsatto *w =
        MACRO_DI_CONVERSIONE(
        gtk_builder_get_object(builder,"nome_widget"));
```

- Le macro di conversione del **GTK+ consistono** nel nome del **widget** di arrivo tutto in maiuscolo
- **Conversione inversa**:

```
GObject *w =
                G_OBJECT(TipoWidgetEsatto *we);
```

Esempio

- Supponiamo di voler accedere ad un bottone (**GtkButton**) e di doverlo passare ad una funzione che prende in ingresso un oggetto di tipo **GtkButton**
- Dobbiamo adottare la soluzione:

```
GtkButton *bottone =
    GTK_BUTTON(gtk_builder_get_object(builder,
                                      "nome_widget"));
```

Attenzione

- Se si accede ad un widget con un nome sbagliato (inesistente), non c'è nessuna segnalazione di errore
- Però la funzione gtk_builder_get_object(...) ritorna un valore inconsistente!
- Fortunatamente la libreria GTK+ è piena di assertion non bloccanti che incominciano a segnalare anomalie sul contenuto del widget
- Tipicamente il messaggio è che il widget non è del tipo atteso

Far apparire/sparire un widget

- Ci manca l'ultimo dettaglio
- Per far **apparire** o **sparire** un **widget,** con tutti i **widget** che contiene, basta invocare rispettivamente le funzioni:

  gtk_widget_show_all o

  gtk_widget_hide
- Entrambe prendono in ingresso un puntatore a **GtkWidget generico**

Procediamo

- Con tutte queste informazioni siamo in grado di scrivere l'**handler** dell'attivazione della voce di menù **Add** affinché faccia apparire la finestra di dialogo
- **Problema**: l'**handler** deve accedere all'oggetto **GtkBuilder** inizializzato dalla funzione **gtk_builder_new** nel **main**
  - Una soluzione semplice è definire il puntatore **builder** all'oggetto **GtkBuilder** a livello di file (meglio se con collegamento interno – dichiarazione **static**)

Codice - GtkBuilder

```
static GtkBuilder *builder; //togliere da main
extern "C"  void handler_activate_Add
                (GtkMenuItem *menuitem,
                 gpointer user_data)
{ GtkWidget *ins_dialog =
    GTK_WIDGET(gtk_builder_get_object(builder,
      "dialog1") );
   gtk_widget_show_all(ins_dialog) ;
}
```

- Compilare e verificare

Handler bottoni

- Per i bottoni (**GtkButton**) ci interessa l'**handler** del segnale **clicked**, che viene emesso dal bottone se si preme e si rilascia il pulsante sinistro del mouse quando il puntatore è sul bottone stesso
- Andiamo a cercare il prototipo dell'**handler** per il segnale **clicked** sul manuale

```
void user_function( GtkButton *button,
                    gpointer data )
```

Handler Cancel

- L'**handler** del bottone **Cancel** è molto semplice, perché deve solo chiudere la finestra di dialogo
- Deve quindi invocare la funzione **gtk_widget_hide** passandole come parametro il **widget** della finestra
- Una prima soluzione è usare di nuovo la **get_widget** per ottenere il **widget** corrispondente alla finestra ed invocarvi sopra la funzione **gtk_widget_hide**

Soluzione generale (1)

Esiste una soluzione più generale

- *Ipotesi*: abbiamo più bottoni **Cancel** che devono soltanto chiudere la finestra a cui appartengono
  - Possiamo scrivere un **handler generico** che risale alla finestra a cui appartiene il bottone che ha emesso il segnale e la chiude
  - Lo associeremo ad ogni bottone **Cancel** per il segnale **clicked**
- L'identificazione del bottone che ha emesso il segnale è data dal primo argomento dell'**handler,** che punta al **button** che emette il segnale per cui l'**handler** è eseguito

Soluzione generale (2)

- Per risalire alla finestra si può invece usare la funzione **gtk_widget_get_toplevel**
  - Prende in ingresso un puntatore a **widget GtkWidget generico**
  - Restituisce il puntatore alla finestra in cui è contenuto il **widget** passato in ingresso
- Scegliamo quindi la strada che preferiamo, implementiamo l'**handler** del bottone **Cancel** ed agganciamolo al relativo segnale **clicked**
- Ricompiliamo ed eseguiamo per controllare che tutto funzioni come atteso

Codice

```
extern "C" void
nascondi_finestra_madre( GtkButton *button,
                         gpointer   data )
{
   gtk_widget_hide( gtk_widget_get_toplevel(
                       GTK_WIDGET(button))) ;
}
```

- *Soluzione in progetto_GUI/GSeq.cc*

Inserimento dell'elemento

- Ci manca ora l'ultimo passo, ossia l'effettivo inserimento dell'elemento alla pressione del bottone **OK**
- Ovviamente si tratta di scrivere un **handler** per il segnale **clicked** del bottone **OK**
- Ma per scrivere l'**handler** abbiamo bisogno di leggere il contenuto di due tipi di **widget**: i **radio button** ed il **textview**

Stato radio button

- Ci serve conoscere lo stato del **radio button**
- Il tipo **radio button** è derivato dal tipo **toggle button**, da cui eredita la funzione **gtk_toggle_button_get_active** che ritorna vero se il bottone è attivo (falso altrimenti)
  - Derivazioni visibili nella *Object Hierarchy* nel manuale
- Possiamo utilizzare tale funzione per scoprire **quale dei bottoni è attivo** e sapere che tipo di inserimento effettuare

Contenuto text view

- Il **widget Text View** memorizza in un oggetto chiamato **GtkTextBuffer** il contenuto inserito dall'utente nella **text view**
- Praticamente questi **buffer** sono oggetti a parte, per cui uno stesso **buffer** può essere per esempio associato a **text view** differenti

Accesso al buffer

- Per accedere al buffer a cui è associata una **text view** possiamo utilizzare la funzione **gtk_text_view_get_buffer**
  - Prende in ingresso un puntatore a **GtkTextView**
  - Restituisce un puntatore al **buffer GtkTextBuffer**
- Per ottenere poi il testo contenuto nel **buffer** sotto forma di stringa bisogna invocare l'ulteriore funzione **gtk_text_buffer_get_text**

Iteratori (1)

- C'è però un'ultima complicazione: bisogna scegliere la porzione del **buffer** da copiare nella stringa
- Per farlo si utilizzano gli **iteratori**, che sono in pratica dei puntatori ai caratteri dentro il **buffer**
- Per ottenere tutto il contenuto del **buffer**, dobbiamo quindi crearci due **iteratori**, uno che punti al primo ed uno che punti all'ultimo elemento del **buffer**

Iteratori (2)

- Siccome il codice per manipolare gli iteratori implica ulteriori dettagli, senza dilungarci in altre spiegazioni rimandiamo direttamente al codice della funzione *insert_handler* in *progetto_GUI/GSeq.cc*
  - *Da associare al segnale clicked del bottone OK*
- Se non si vuole leggere il codice di tutta la funzione, considerare che questa parte è evidenziata da due apposite righe di commento al suo inizio ed alla sua fine

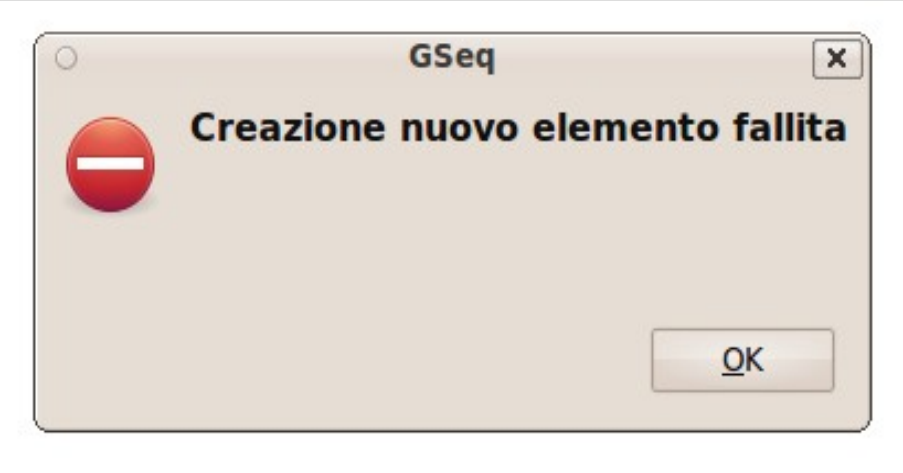
Creazione elemento

- Letto il contenuto del **buffer**, possiamo utilizzarlo per creare il nuovo elemento della sequenza
- Ma, come vediamo, la nuova funzione **crea_elem** (definita in *manip_stampa.cc in progetto_GUI*) prende in ingresso un **istream**
  - Nel vecchio programma l'inserimento di una sequenza avveniva interattivamente nel main parola per parola: ora non è possibile
- Anche un **istringstream** è un tipo di **istream**
- Pertanto possiamo creare un **istringstream** inizializzato con la stringa letta dal **buffer** e passarlo alla funzione **crea_elem**

Finestra di errore (1)

In caso di fallimento dell'inserimento dobbiamo far apparire una **finestra di errore**

- Conosciamo il meccanismo
- Esempi



Finestra di errore (2)

Si potrebbe pensare ad una generica **finestra di errore** composta da un'**immagine** e **due label che cambiano a seconda dell'errore**

- Per gestire la finestra di errore (chiamata **error_alert**) in *progetto_GUI/GSeq.cc* si usa una generica funzione privata (static) **mostra_error_alert** che, prima di farla apparire, modifica il contenuto delle due **label**
  - Le inizializza con le due stringhe passate in ingresso
  - Inoltre fa apparire il testo della prima label più grande dell'altro

Finestra di errore (3)

- Disegnare la finestra di errore con **Glade** senza curarsi del testo delle due label
  - Si tratta sempre di una **finestra di dialogo**
  - *Number of items=1* nella *dialog-action-area*
  - Chiamare le label *label_primaria* e *label_secondaria*
- Guardare il codice della funzione **mostra_error_alert** nel programma di riferimento *progetto_GUI/GSeq.cc*
  - Si utilizzano due funzioni per modificare il testo delle label: **gtk_label_set_text** e **gtk_label_set_markup**

Testo semplice e formattato

- Funzione **gtk_label_set_text** per settare il testo dell'etichetta più in basso (*label_secondaria*) con una stringa che non contenga indicazioni di formattazione
- Funzione **gtk_label_set_markup** per settare il testo dell'etichetta più in alto (*label_primaria*) con un font più grande e in grassetto
  - Questa funzione permette di inserire il testo di una label usando un semplice linguaggio di formattazione (**markup**), che a sua volta consente, per esempio, di scegliere il font desiderato

Come procedere

- Non è ovviamente necessario conoscere tutti i dettagli della **formattazione** del testo nelle **label**
- Per continuare con il programma, potete semplicemente copiare la formattazione utilizzata in *progetto_GUI/GSeq.cc*

Inserimento

- Infine, bisogna **inserire l'elemento nella sequenza**
- Quindi la funzione **handler_inserisci** deve poter accedere alla sequenza
  - Per poterla passare alla funzione **inserisci**
- Un modo semplice per ottenere questo risultato è definire la sequenza come **variabile** con **visibilità di file**
  - Meglio se con collegamento interno (static)
  - Prima era definita nel **main**

Aggiornamento text view

Si deve infine aggiornare il contenuto della **text view** che mostra il contenuto della sequenza nella finestra principale

- Bisogna ottenerne il **buffer** nel solito modo
- Per settare il nuovo contenuto del buffer si usa la funzione **gtk_text_buffer_set_text** a cui si passa il **buffer** da modificare, la stringa con il nuovo contenuto, e la lunghezza del testo
  - Se lunghezza = -1 si ferma quando incontra NULL
- Il problema è costruire tale stringa, che contiene praticamente la stampa della sequenza

Ostringstream (1)

- Noi già disponiamo della funzione **stampa_elem**, che manda l'output su un **ostream**
  - Finora la abbiamo invocata passandole **cout** oppure un **ofstream**
- Esiste un altro tipo di **ostream** molto comodo, chiamato **ostringstream**
  - Per definire un oggetto di tipo **ostringstream**:

    ostringstream oss ;

Ostringstream (2)

- Quello che viene mandato in ingresso ad un **ostringstream** non viene stampato su **stdout** né scritto in un file, ma bensì memorizzato in una variabile interna di tipo stringa
- Es.:

  oss<<"prova "<<13 ;

  Fa finire nella variabile interna la stringa “prova 13”
- Per accedere alla stringa, possiamo utilizzare la chiamata composta **oss.str().c_str()** che ritorna un puntatore ad un **array di caratteri** contenente tale stringa

Ostringstream (3)

- Infine, per svuotare la stringa interna : oss.str(""); oss.clear() ;
- Possiamo quindi mandare l'output di **stampa_elem** ad un **ostringstream** e usare poi la stringa interna dell'**ostringstream** per aggiornare il contenuto della **text view**
- Codice *progetto_GUI/GSeq.cc*

Barra di stato

- Ricordarsi che bisogna aggiornare anche la barra di stato
- Significa aggiornare le label
  - label_numelem
  - label_ordinata
  - label_tipochiave

Completamento interfaccia

- Ora disponete di tutte le informazioni che vi servono per completare l'interfaccia grafica
  - A parte la generazione dell'istogramma che sarà l'ultimo argomento trattato

Integrazione conoscenze (1)

- Non abbiamo ovviamente visto tutte le proprietà dei **widget**
- C'è una soluzione molto semplice per estendere le proprie conoscenze
  - Per ciascun **widget**, quasi ogni elemento della (sotto)finestra **Properties di Glade** ha un **tooltip** (suggerimento) visualizzato se ci si ferma sopra l'elemento col mouse

Integrazione conoscenze (2)

Per ciascuno dei **widget** inseriti nell'interfaccia del programma scritto finora:

- Si può leggere il suggerimento fornito da **Glade** per ciascuna delle sue proprietà
- Assicurarsi di averlo capito
- Se opportuno, settare la relativa proprietà al valore più adatto per migliorare la qualità dell'interfaccia