

Sistemi per l'elaborazione dell'informazione 2

Prologo

Oltre alla descrizione delle varie esercitazioni da eseguire in laboratorio (o per conto vostro), sul sito del corso trovate anche le corrispondenti soluzioni. Resistete alla tentazione di sbirciare fino a che non siete soddisfatti della soluzione che avete trovato per conto vostro. Solo allora confrontate il vostro risultato con quello pubblicato e verificate se vi sia una differenza nell'efficienza delle implementazioni.

SWT

Lo scopo di questa esercitazione è la realizzazione di interfacce grafiche tramite SWT, e comprende la creazione di widget, la loro sistemazione nell'interfaccia tramite l'uso di dispositori e l'utilizzo dei disegni *mediatore* e *comando* per ottenere delle interfacce relativamente complesse. Va notato come il collegamento dei widget con il codice da eseguire quando l'utente interagisce con essi (per intenderci, l'utilizzo dei derivati di **Listener**) implementa un altro disegno (detto *osservatore/osservabile*) che studieremo in dettaglio in una delle prossime lezioni.

Configurazione

Per poter utilizzare SWT, Eclipse ha bisogno di sapere dove sono i jar associati. Una volta creato il progetto, andate in Project/Properties/Java Build Path/Libraries, selezionate “Add JARs...” e aggiungete i due jar

```
/programmi/eclipse/plugins/org.eclipse.swt.gtk_3.1.1/ws/gtk/swt.jar  
/programmi/eclipse/plugins/org.eclipse.swt.gtk_3.1.1/ws/swt-pi.jar
```

Cliccando sui triangoli accanto ai jar appena aggiunti, verranno evidenziate diverse voci, tra cui “Javadoc location”: selezionatela, e puntatela alla radice della documentazione di SW

```
/programmi/eclipse/eclipse-doc  
http://download.eclipse.org/eclipse/downloads/documentation/2.0/html/  
plugins/org.eclipse.platform.doc.isv/reference/api/
```

in modo che Shift-F2 funzioni correttamente. In ogni caso, potete sempre accedere a documentazione sui widget aprendo l'aiuto di Eclipse e utilizzando la ricerca testuale (provate, ad esempio, con “swt GridLayout”).

Creazione di controlli

Provate a creare controlli di testo, etichette, e così via, utilizzando come codice di base quello disponibile sul sito e mostrato a lezione. Evitate di utilizzare i metodi di posizionamento assoluto: create fin dall'inizio un dispositore (per esempio, un **RowLayout**) e assegnatelo alla vostra **Shell**. In particolare, realizzate le interfacce descritte nei paragrafi che seguono.

Gestione di eventi

Agganciate ai vostri controlli degli osservatori (in interfaccese, *listener*) che fanno interagire i controlli tra di loro.

- Create un composto con un controllo testuale affiancato da un pulsante **Button** con etichetta “Reset” che cancella il contenuto del controllo testuale (dovete usare il metodo `setText()`). Notate che, affinché sia possibile invocare all’interno dell’osservatore i metodi di **Text**, l’istanza di quest’ultimo deve essere dichiarata **final**.
- Create un composto con un controllo testuale, un pulsante “Aggiungi” e una lista sottostante (potete partire dall’esempio mostrato a lezione). Alla pressione del tasto “Aggiungi”, il contenuto corrente del controllo testuale deve essere aggiunto alla lista. Se nella lista viene effettuata una scelta, l’etichetta dell’elemento scelto deve sostituire il contenuto del controllo di testo (dovete usare il metodo `add()` di **List**).
- Create due **Slider** agganciati in modo che le modifiche al primo siano riflesse nel secondo, ma non viceversa. Per farlo, dovete agganciare un **SelectionListener** al primo **Slider** che recupera con `getSelection()` il valore corrente e lo imposta nel secondo **Slider**.

Dispositori

Sperimentate i dispositori di base **FillLayout** e **RowLayout**. Una volta presa familiarità, provate ad utilizzare un **GridLayout**. Il principio di un **GridLayout** è molto semplice: c’è un numero di colonne, fissato al momento della costruzione del dispositore. Ogni composto occupa un certo numero di righe e colonne (d’ufficio, una). Inoltre, potete associare esplicitamente a ogni composto un oggetto **GridData** che modifica la politica d’ufficio tramite il metodo `setLayoutData()`: potete, ad esempio, fare occupare a un composto più colonne (modificando il campo `horizontalSpan`), chiedere che il composto venga espanso fino a occupare tutto lo spazio possibile (impostando `horizontalAlignment = GridData.FILL` e `grabExcessHorizontalSpace = true`) e così via. Utilizzate la documentazione in linea di Eclipse per provare le varie opzioni.

Sia **GridLayout** che **GridData** dispongono di diversi costruttori che permettono, nella stragrande maggioranza dei casi, di non dover creare una variabile locale, modificarla, e passarla a `setLayoutData()`, ma piuttosto di fare tutto al volo (per esempio, `setLayoutData(new GridData(GridData.FILL_HORIZONTAL))`). Leggete con cura la documentazione disponibile in linea e la documentazione Javadoc per scegliere che metodo utilizzare.

Nota bene: in un **Composite** il metodo `setLayoutData()` modifica le caratteristiche di disposizione *all’interno del composto genitore*, mentre `setLayout()` sceglie il dispositore per gli elementi contenuti nel **Composite**. Questo vi permette di utilizzare diversi dispositori per zone diverse della finestra. È consigliabile, se tentate delle disposizioni complicate, formalizzarle prima su carta.

Sfida!

Spesso le esercitazioni termineranno con un esercizio ironicamente definito *Sfida!*, apparentemente destinato ai più temerari, ma che in realtà è tranquillamente risolvibile una volta che gli argomenti spiegati a lezione sono stati appresi. Questi esercizi richiederanno tipicamente di *riutilizzare* il codice visto a lezione o scritto durante il laboratorio, evitando ove possibile di andare a modificare quanto già scritto in favore di un utilizzo corretto dei disegni progettuali.

La sfida di questa lezione richiede di riconsiderare i disegni *mediatore* e *comando* unitamente all’interfaccia grafica **UpDownShell**. In particolare:

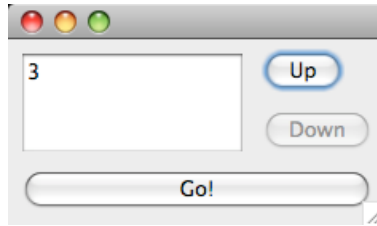


Figura 1: L'interfaccia da utilizzare nella classe `EnhancedUpDownShell`.

- Realizzate una classe `EnhancedMediator` che effettui le stesse operazioni della classe `Mediator` vista a lezione, permettendo però di impostare un valore minimo per il range di interi specificabili.
- Realizzate altresì una classe `EnhancedUpDownShell` che permetta di lavorare con `EnhancedMediator` e verificate come sia possibile utilizzare questa nuova classe nel codice visto a lezione al prezzo di inserire modifiche minime nella sola classe di test.
- Modificate l'interfaccia grafica di `EnhancedUpDownShell` sfruttando un `GridLayout` in modo da ottenere l'effetto illustrato in Figura 1 (notate che le dimensioni della casella di testo e del pulsante *Go!* vengono automaticamente modificate quando la finestra viene ridimensionata).
- Realizzate una classe `DrawPolygonCommand` che implementi l'interfaccia `Command` in modo che quando si invoca `execute(n)` venga visualizzata una nuova finestra (legata quindi a una nuova istanza di `Shell`) nel cui contenuto risulti disegnato un poligono regolare di `n` lati. Non spaventatevi se non sapete come si fa a “disegnare”: basta inserire nella shell un'istanza di `Canvas` collegata con un `PaintListener`, in cui venga implementato il metodo `PaintControl`. Se chiamiamo `e` l'argomento di questo metodo, `e.gc` fornisce un oggetto, detto *contesto grafico*, su cui è possibile invocare il metodo `drawLine` che accetta come argomenti quattro interi che definiscono rispettivamente ascissa e ordinata degli estremi di un segmento. Ricordate che l'origine è posta in corrispondenza dell'angolo in alto a sinistra di ogni finestra. Infine, rispolverate le conoscenze di base della trigonometria. In ogni caso, niente panico e ricordate che la documentazione è sempre vostra amica.
- Mettete assieme il tutto in modo che un'istanza di `EnhancedUpDownShell`, caratterizzata da un range di valori ammissibili tra 3 e 9, visualizzi il corrispondente poligono regolare in una nuova finestra ogni volta che viene premuto il pulsante *Go!*.