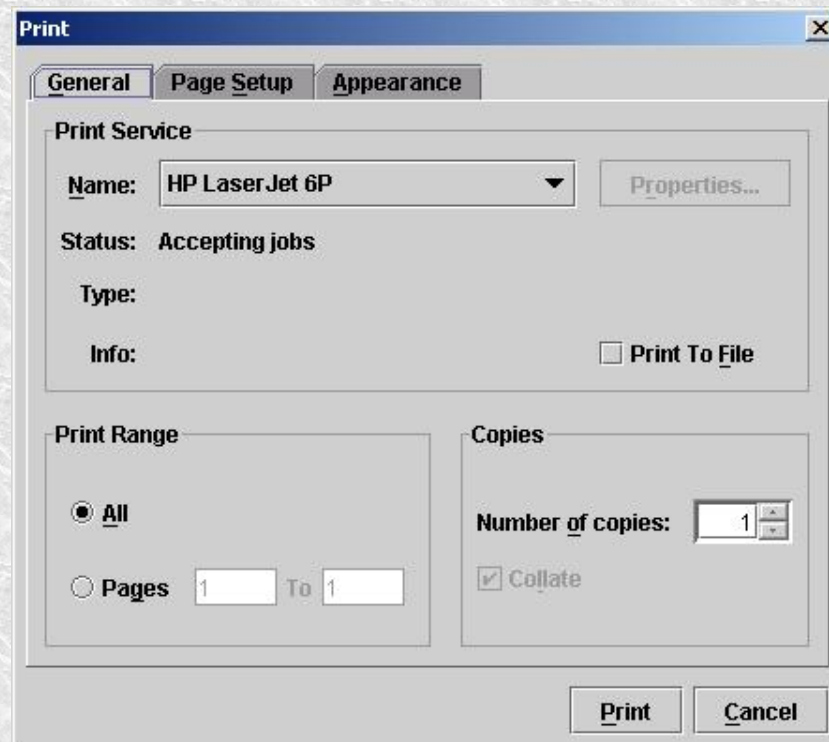


Laboratorio di IUM

Lezione 8

L'AWT che serve



Cosa impareremo oggi?

- L'API per la stampa
- La clipboard di sistema
- Il Drag'n'Drop

L'API per la stampa (1)

- Il kit di sviluppo originale di Java non forniva alcun supporto per la stampa
- Dalle applet era impossibile stampare e dalle applicazioni era possibile solo con librerie esterne
- Con la versione 1.1 è stato introdotto un supporto minimale che in pratica era assolutamente inutile
- Dalla versione 1.2 fino alla 1.5 il supporto per la stampa è stato sempre più migliorato ed integrato con l'API *Java2D* diventando uno strumento potente e flessibile

L'API per la stampa (2)

- L'integrazione con *Java2D* ha reso il processo di stampa “semplice” come il disegno su un componente visuale
- Nonostante ciò è bene chiarire che realizzare un software di stampa è un lavoro complesso e difficile
 - spesso è meglio appoggiarsi ad API progettate ad-hoc come JasperReport (<http://jasperforge.org/sf/projects/jasperreports>)
- Per generare una stampa bisogna effettuare due operazioni:
 - fornire un oggetto che implementa l'interfaccia *Printable*
 - avviare un “lavoro di stampa”

L'interfaccia Printable (1)

- L'interfaccia *Printable* rappresenta qualsiasi oggetto che deve essere “stampabile”
- Implementare tale interfaccia è molto semplice basta implementare il metodo
 - `int print(Graphics g, PageFormat format, int page):`
descrive le operazioni di stampa dell'oggetto
- Tale metodo viene chiamato ogni volta che il motore di stampa deve “formattare” una pagina

L'interfaccia Printable (2)

- La stampa avviene “disegnando” sul contesto grafico *g*
- *format* indica invece le dimensioni della carta ed i valori dei margini
- *page* specifica la pagina da stampare (partendo da zero)
- Il valore restituito può essere `PAGE_EXISTS` oppure `NO_SUCH_PAGE` ed indica se ci sono o meno altre pagine da stampare

Avviare un lavoro di stampa (1)

- Per stampare bisogna utilizzare la classe *PrinterJob* ed:
 - ottenere la classe *PrinterJob* tramite il metodo statico
 - `PrinterJob getPrinterJob()`
 - impostare l'oggetto *Printable* nella *PrinterJob* tramite i metodi
 - `void setPrintable(Printable painter)`
 - `void setPrintable(Printable painter, PageFormat format)`
 - visualizzare la finestra di dialogo per la stampa con i metodi
 - `boolean printDialog()`
 - `boolean printDialog(PrintRequestAttributeSet attributes)`
 - avviare la stampa con i metodi
 - `void print()`, `void print(PrintRequestAttributeSet attributes)`

Avviare un lavoro di stampa (2)

Esempio

PrinterJobSample

Avviare un lavoro di stampa (3)

- Osservazione:
 - il lavoro di stampa non ha alcun modo per conoscere il conteggio esatto delle pagine
 - per tale motivo nella finestra di stampa si visualizza un intervallo “Pagine da 1 a 1”
 - a breve vedremo come risolvere il problema tramite l'oggetto *Book*

La classe PageFormat (1)

- La classe *PageFormat* descrive dimensione ed orientamento della pagina da stampare
- Tutte le dimensioni sono misurate in “punti”
- Un *punto* è 1/72 di pollice (25,4 mm)
 - per esempio il formato A4 è circa 595x842 punti
- Il sistema di stampa Java utilizza il punto per due ragioni:
 - le dimensioni ed i margini della carta sono misurate in punti
 - il punto è l'unità predefinita per tutti i contesti grafici di stampa

La classe PageFormat (2)

- Costruttore
 - *PageFormat()*
- Metodi utili
 - double *getWidth()*, double *getHeight()*: restituisce le dimensioni della pagina
 - double *getImageableX()*, double *getImageableY()*: restituisce le coordinate (x, y) del punto in alto a sinistra dell'area stampabile
 - double *getImageableWidth()*, double *getImageableHeight()*: restituisce le dimensioni dell'area stampabile
 - int *getOrientation()*, void *setOrientation(int orientation)*: restituisce/imposta l'orientamento della pagina

Impostazione della pagina (1)

- Per impostare gli attributi della pagina senza impostare altri attributi (e senza stampare) si possono utilizzare i metodi
 - PageFormat *pageDialog*(PageFormat page),
 - PageFormat *pageDialog*(PrintRequestAttributeSet attributes): visualizzano una finestra di dialogo per impostare gli attributi della pagina
- Mentre per ottenere gli attributi di default si usa il metodo
 - PageFormat *defaultPage*(): crea un nuovo *PageFormat* con gli attributi di default

Impostazione della pagina (2)

Esempio

PageSetupSample

Stampa di più pagine (1)

- Solitamente quando si stampa non si utilizza il semplice oggetto che implementa *Printable*, ma un oggetto che implementa *Pageable*
- *Pageable* rappresenta un insieme di pagine da stampare
- La piattaforma Java fornisce la classe *Book* che implementa *Pageable*

Stampa di più pagine (2)

- Un *Book* è un libro costituito da paragrafi ognuno dei quali è un oggetto *Printable*
- Ogni *Printable* può avere un diverso formato di pagina
- Per stampare un *Book* bisogna impostarlo nel lavoro di stampa tramite il metodo di *PrinterJob*
 - void *setPageable*(Pageable document): imposta l'oggetto *Pageable* nel lavoro di stampa

Stampa di più pagine (3)

- Costruttore
 - *Book()*
- Metodi utili
 - void *append(Printable painter, PageFormat page)*: aggiunge un *Printable* (di una pagina) alla fine del *Book*
 - void *append(Printable painter, PageFormat page, int numPages)*: aggiunge un *Printable* (di *numPages* pagine) alla fine del *Book*
 - void *setPage(int pageIndex, Printable painter, PageFormat page)*: imposta un *Printable*
- Continua...

Stampa di più pagine (4)

- Segue
 - int *getNumberOfPages()*: restituisce il numero di pagine di questo Book
 - PageFormat *getPageFormat*(int pageIndex): restituisce il formato di una pagina
 - Printable *getPrintable*(int pageIndex): restituisce un *Printable*
- Nota: adesso il *PrinterJob* sa quante pagine deve stampare

Stampa di più pagine (5)

Esempio

MultiPagePrintSample

Altri argomenti

- Anteprima di stampa
 - Servizi di stampa
 - Servizi di stampa di flusso
 - Attributi di stampa
-
- Non verranno trattati ma sono molto utili ed interessanti (se ne consiglia lo studio individuale)

La clipboard di sistema (1)

- Uno dei meccanismi più utili di tutte le moderne GUI è sicuramente il copia&incolla
- Questa operazione è implementata tramite i cosiddetti appunti di sistema e tutto il lavoro è svolto “gratuitamente” dal SO sottostante
- JDK 1.0 non aveva alcun supporto a tale funzionalità
- A partire dalla versione 1.1 tale supporto è stato gradualmente inserito e migliorato

La clipboard di sistema (2)

- Adesso è possibile trasferire:
 - testo, immagini, elenchi di file, ecc. tra un programma Java ed un programma nativo
 - oggetti serializzati e remoti tra due programmi Java eseguiti su due JVM diverse
 - qualsiasi oggetto all'interno della stessa JVM
- Noi ci occuperemo solo del primo caso (che è ovviamente il più interessante)

La classe *DataFlavor* (1)

- La classe *DataFlavor* descrive i tipi di dato che possono essere copiati negli appunti
- Solo i tipi definiti da *DataFlavor* possono essere copiati
- Attualmente la classe *DataFlavor* definisce una serie di oggetti statici per diversi tipi di dato

La classe DataFlavor (2)

- In particolare:
 - l'oggetto *DataFlavor.imageFlavor* rappresenta un'immagine
 - l'oggetto *DataFlavor.javaFileListFlavor* rappresenta una lista di file
 - la stringa *DataFlavor.javaJVMLocalObjectMimeType* rappresenta un oggetto da trasferire nella stessa JVM
 - la stringa *DataFlavor.javaRemoteObjectMimeType* rappresenta un oggetto da trasferire tra JVM differenti
 - la stringa *DataFlavor.javaSerializedObjectMimeType* rappresenta un oggetto serializzabile
 - l'oggetto *DataFlavor.stringFlavor* rappresenta una stringa

L'interfaccia Transferable (1)

- Se un oggetto vuole essere “trasferito” da un'applicazione ad un'altra deve implementare l'interfaccia *Transferable*
- Ovvero deve implementare i metodi
 - Object *getTransferData(DataFlavor flavor)*: restituisce l'oggetto che rappresenta il dato da trasferire
 - DataFlavor[] *getTransferDataFlavors()*: restituisce un array di oggetti *DataFlavor* che indicano che tipi di dato possono essere forniti
 - boolean *isDataFlavorSupported(DataFlavor flavor)*: comunica se questo tipo di dato è supportato

L'interfaccia Transferable (2)

- Java fornisce un'implementazione di *Transferable* per le stringhe
- Tale implementazione è la classe *StringSelection*
- Costruttore
 - *StringSelection*(String data)

La classe Clipboard

- La classe *Clipboard* rappresenta gli appunti di sistema
- Per ottenere un'istanza di *Clipboard* bisogna ricorrere alla classe *Toolkit*
 - *Clipboard* *getSystemClipboard()*: restituisce la clipboard di sistema
- Metodi utili
 - *Transferable* *getContents*(Object requestor), void *setContents*(Transferable contents, ClipboardOwner owner): restituisce/imposta i dati nella clipboard

La clipboard di sistema (3)

Esempi

TextTransferSample
ImageTransferSample

Il Drag'n'Drop

- Gli appunti di sistema fungono da intermediario nelle operazioni di copia&incolla tra due programmi
- Il *Drag'n'Drop* (letteralmente “trascina e rilascia”) elimina l'intermediario e consente la comunicazione diretta tra due programmi
- Java fornisce un supporto elementare ma sufficiente per le operazioni di drag'n'drop

Operazioni di rilascio (1)

- Per prima cosa vediamo come un programma Java può diventare una destinazione di operazioni di rilascio
- Tutti i componenti AWT (e quindi gli *Swing*) possono essere una destinazione di operazioni di rilascio
- La prima cosa da fare è creare un oggetto *DropTarget*, ovvero il target di un'operazione di rilascio

Operazioni di rilascio (2)

- Per lavorare un *DropTarget* deve essere associato al componente destinazione ed ad un listener che ascolta le operazioni di rilascio
- Costruttori
 - *DropTarget*(Component c, DropTargetListener dtl)
 - *DropTarget*(Component c, int ops, DropTargetListener dtl)
 - *DropTarget*(Component c, int ops, DropTargetListener dtl, boolean act)
 - *DropTarget*(Component c, int ops, DropTargetListener dtl, boolean act, FlavorMap fm)

Operazioni di rilascio (3)

- Metodi utili
 - void *setActive*(boolean isActive): attiva/disattiva la destinazione
 - void *setDefaultActions*(int ops): imposta le operazioni di rilascio consentite sulla destinazione, secondo le costanti della classe *DnDConstants*
 - ACTION_COPY: azione di copia
 - ACTION_MOVE: azione di spostamento
 - ACTION_COPY_OR_MOVE: azione di copia o spostamento
 - ACTION_LINK: azione di creazione di un link simbolico
 - ACTION_NONE: nessuna azione

Operazioni di rilascio (4)

- A questo punto bisogna implementare l'interfaccia *DropTargetListener*, ovvero implementare i metodi
 - void *dragEnter*(DropTargetDragEvent dtde)
 - void *dragOver*(DropTargetDragEvent dtde)
 - void *dragExit*(DropTargetEvent dte): chiamati quando il mouse entra/si muove/esce da una zona “rilasciabile”
 - void *drop*(DropTargetDropEvent dtde): chiamato quando l'utente rilascia il tasto del mouse
 - void *dropActionChanged*(DropTargetDragEvent dtde): chiamato se l'utente modifica il gesto che ha iniziato l'azione di drag (pressione/rilascio di tasti CTRL, ALT, MELE, ecc.)

Operazioni di rilascio (5)

Esempio

DropSample

Operazioni di trascinamento (1)

- Adesso vediamo come un programma Java può diventare un'origine di operazioni di trascinamento
- Per rendere un componente un'origine di operazioni di trascinamento bisogna:
 - ottenere un oggetto *DragSource* tramite il metodo statico
 - *DragSource getDefaultDragSource()*: restituisce un'origine di drag associato con il SO sottostante
 - configurare l'origine con il metodo
 - *DragGestureRecognizer createDefaultDragGestureRecognizer(Component c, int actions, DragGestureListener dgl)*: crea un “riconoscitore” di operazioni di drag, associato al componente *c*

Operazioni di trascinamento (2)

- A questo punto bisogna implementare l'interfaccia *DragGestureListener*, ovvero il metodo
 - void *dragGestureRecognized*(DragGestureEvent dge): invocato quando il SO sottostante intercetta l'inizio di un'operazione di drag
- All'interno di questo metodo il programmatore deve:
 - definire l'oggetto *Transferable* target dell'operazione di drag
 - lanciare il metodo di *DragGestureEvent*
 - void *startDrag*(Cursor dragCursor, Transferable transferable, DragSourceListener dsl): avvia l'operazione di drag

Operazioni di trascinamento (3)

- Infine bisogna implementare l'interfaccia *DragSourceListener*, ovvero i metodi
 - void *dragEnter*(DragSourceDragEvent dsde)
 - void *dragOver*(DragSourceDragEvent dsde)
 - void *dragExit*(DragSourceEvent dse): chiamati quando il mouse entra/si muove/esce da una zona “trascinabile”
 - void *dragDropEnd*(DragSourceDropEvent dsde): chiamato quando l'utente rilascia il tasto del mouse
 - void *dropActionChanged*(DragSourceDragEvent dsde): chiamato se l'utente modifica il gesto che ha iniziato l'azione di drag (pressione/rilascio di tasti CTRL, ALT, MELA, ecc.)

Operazioni di trascinamento (4)

Esempio

DragSample