LPR 2005/ 2006
Lezione n.9

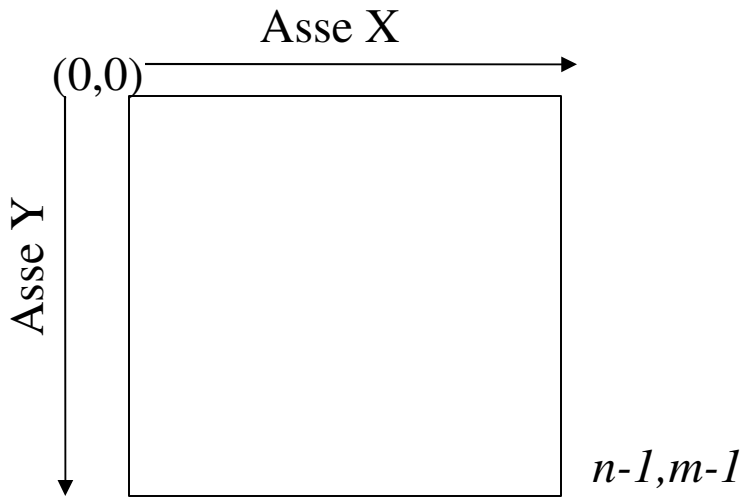
- Elementi di grafica per lo svolgimento del progetto
- Double Buffering
- Caricamento delle immagini
- Gestione degli eventi

JAVA 2D PROGRAMMING

- *Schermo* : Matrice rettangolare composta di picture elements (*pixels*) di dimensione *n, m*

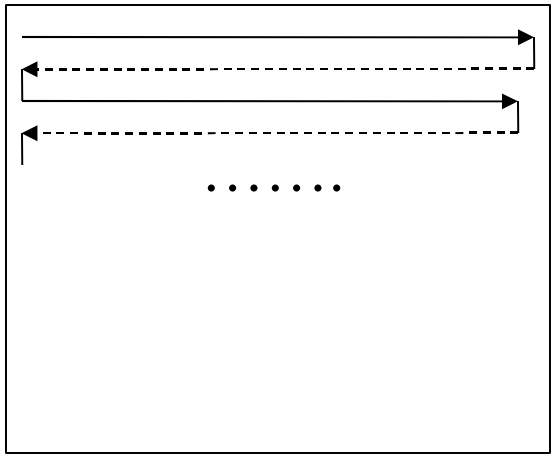
- ogni pixel memorizza un colore usato per visualizzare immagini o sfondi

- le immagini possono essere posizionate rispetto ad un sistema di riferimento cartesiano con l'origine in alto a sinistra



JAVA 2D PROGRAMMING

- *Risoluzione :* definita da altezza, profondità, *bit depth* dello schermo
- *Bit depth:* indica quanta memoria è utilizzata per memorizzare un pixel
- *Rendering:* Visualizzazione di *un frame*. Il frame viene prelevato RAM del video e visualizzato



JAVA 2D PROGRAMMING

Approcci per la progettazione di applicazioni 2D in JAVA. Per il processo di rendering è possibile utilizzare

- lo stesso meccanismo utilizzato per il rendering delle componenti GUI (bottoni, textbowes, etc…) in finestre AWT o SWING.

- JAVA 1.4introduce la *full screen API*, un insieme di classi progettate appositamente per la progettazione di giochi.

Alcuni vantaggi del secondo approccio:

- esecuzione del gioco in modalità *full screen*
- possibilità di scegliere la risoluzione migliore per un certo tipo di gioco
- *double buffering*

JAVA 2D PROGRAMMING: FULL SCREEN API

Utilizzo della Full Screen API:

Passo 1: Ottenere un riferimento all'ambiente grafico presente sull'host su cui è in esecuzione l'applicazione. Conoscere quanti schermi sono collegati all'host e le caratteristiche di ognuno di essi

GraphicsEnvironment
gfxEnv=*GraphicsEnvironment.getLocalGraphicsEnvironment*( )

GraphicsDevice [ ] screenList = gfxEnv.*getScreenDevices*( )

GraphicsDevice = Vettore di oggetti che descrivono gli schermi collegati all'host

Se esiste un solo video (vero nella maggior parte dei casi)

GraphicsDevice [ ] defaultDevice = gfxEnv.*getDefaultScreenDevices*( )

JAVA 2D PROGRAMMING: FULL SCREEN API

Utilizzo della Full Screen API:

Passo2: Ottenere le caratteristiche del/degli schermi collegati all'host

DisplayMode [ ] displayModes
= *defaultDevice.getDisplayModes*( );

restituisce un vettore che contiene le modalità di visualizzazione di un certo dispositivo video.

Passo3: Confrontare le modalità di visualizzazione richieste dalla applicazione con quelle disponibili sullo schermo collegato. Scegliere la modalità di visualizzazione più idonee per l'applicazione ed impostarle opportunamente.

JAVA 2D PROGRAMMING: FULL SCREEN API

Nel nostro caso non siamo interessati a risoluzioni video particolari. Quindi svilupperemo l'applicazione nell'ipotesi che

- esista un solo schermo collegato all'host
- la configurazione video prescelta sia quella definita di default
- l'area di rendering viene definita all'interno di una finestra *(frame)*

Sotto queste ipotesi:

GraphicsEnvironment gfxEnv =
Graphics Environment.getLoaclGraphicsEnvironment( );

GraphicsDevice defaultDevice=gfxEnv.g*etDefaultScreenDevice*( );

GraphicsConfiguration gfxCon=defaultDevice.*getDefaultConfiguration*( );

JAVA 2D PROGRAMMING: FULL SCREEN API

Creazione di un frame a cui è associata la configurazione grafica individuata

```
class RenderingWindow extends Frame
{private int windowWidth=640;
private int windowHeigth=640;
public RenderingWindow (String title, GraphicsConfiguration gfxCon)
                        throws Exceptions;
    { super(title, gfxCon);
     Color backgroundColor=Color.black;
     setUndecorated(false);
     setIgnoreRepaint(true);
    setSize(windowWidth,windowHeight);
    setVisible(true);}}
```

JAVA 2D PROGRAMMING: FULL SCREEN API

- *super*(title, gfxCon): invoca il costruttore della classe padre (frame) e costruisce la finestra attibuendo il titolo *title* ed associando ad essa la configurazione grafica gfxCon

- *Color* backgroundColor=*Color.black* imposta il valore di backgroud della finestra

- *setUndecorated*(false); per eliminare il titolo, etc. dalla barra della finestra

- *setIgnoreRepaint*(true);

- *setVisible*(true) visualizza la finestra

JAVA 2D PROGRAMMING: IL RENDERING

Allocazione del frame:

```
RenderingWindow rw = new
RenderingWindow (“Guardie&Ladri”,gfxCon);
```

A questo punto è possibile effettuare rendering sul frame allocato.

```
rw.render( )
```

Il metodo render contiene il codice per generare l’animazione sulla finestra di rendering

JAVA 2D PROGRAMMING: IL RENDERING

Approcci al rendering:

- La posizione di tutti gli oggetti della scena viene aggiornata direttamente nella memoria video (problemi flashing, tearing)

- *Double buffering:* vengono utilizzati due buffers
  - il front buffer contiene l'immagine visualizzata attualmente sullo schermo
  - il nuovo frame viene creato in un *back buffer*. Il back buffer viene copiato nel front buffer per la visualizzazione
  - per evitare la copia del back buffer nel front buffer $\Rightarrow$ tecnica del *page flipping* (scambio puntatori front/end buffer*)*

JAVA 2D PROGRAMMING: IL RENDERING

- Per effettuare il rendering con il double buffering in JAVA:

```
createBufferStrategy(2);
BufferStrategy bs = getBufferStrategy( );
Graphics g =bs.getDrawGraphics();
```

- Per disegnare forme geometriche, immagini, scrivere stringhe su g

```
g.setColor(color.black);
g.fillRect(0,0,640,640)
String s="sono una guardi"
g.drawString(s,200,200)
```

- Per visualizzare il frame, quando è stato completamente disegnato:

```
bs.show( )
```

JAVA 2D PROGRAMMING: IMPORTARE LE IMMAGINI

- JAVA 2D contiene diverse classi che consentono di caricare e manipolare immagini di diverso formato (esempio: png, gif,…)

- Sono definite classi con funzionalità diverse

  – *BufferedImage:* memorizza l'immagine nella RAM dell'host e consente la manipolazione diretta dell'immagine

  – *VolatileImage:* l'immagine è memorizzata direttamente nella RAM del video ⇒ non si può manipolare l'immagine, però il disegno dell'immagine sul video è più efficiente

  – *Managed Images:* compromesso tra i due approcci precedenti

JAVA 2D PROGRAMMING: IMPORTARE LE IMMAGINI

Come utilizzare la classe *BufferedImages*

String imageResourceName = "ladro.png";

BufferedImage image = *null;*

URL imageURL = *getClass().getResource(imageResourceName);*

*if (*imageURL ==*null) {System.out.println (*"Non riesco a caricare l'immagine*");*
System.exit(0);}
else { image=*ImageIO.read(imageURL); }*

JAVA 2D PROGRAMMING: RENDERING DELL'IMMAGINI

Per effettuare il *rendering* dell'immagine importata:

```
Random rand = new Random( );
if (image!=null)
{for (int loop=0; loop<100; loop++)
        {int x = rand.nextInt (getWidth(  )),
         int y = rand.nextInt (getHeight(  )),
        g.drawImage(image,x,y, this)
}
```

g.drawImage visualizza image alla posizione x,y. Il quarto parametro deve essere un oggetto di tipo ImageObserver nel caso in cui l'immagine può non essere completamente caricata. Questo non è possibile se si usa la classe *ImageIO*.

JAVA 2D PROGRAMMING: GESTIONE DEGLI EVENTI

- In una applicazione interattiva, come un gioco, è necessario gestire eventi che vengono generati dalla interazione con l'utente

- Eventi possibili

  - *pressione di un tasto* (ad esempio freccia in verso l'alto, per spostare il ladro controllato dal giocatore locale, tasto ESC per uscire dal gioco,…)
  - *click del mouse* (ad esempio il mouse può indicare la posizione in cui va spostato un giocatore, per spostamenti lunghi…)
  - chiusura o iconizzazione di finestre

JAVA 2D PROGRAMMING: GESTIONE DEGLI EVENTI

- gli eventi si verificano in *modo asincrono* rispetto all'esecuzione del programma
- L'applicazione non può controllare continuamente l'eventuale verificarsi di un evento (*polling*)
- L'evento deve essere gestito in un thread di controllo separato rispetto al main thread della applicazione
- Quando l'evento si verifica viene gestito in maniera trasparente rispetto all'applicazione

JAVA 2D PROGRAMMING: GESTIONE DEGLI EVENTI

Gestione degli eventi in JAVA:

- definizione di un *gestore di evento* GE per ogni evento che si può verificare all'interno di un *frame F*

- registrazione di un *listener* per ogni tipo di evento che si può verificare all'interno di un *frame F*. Il listener intercetta l'evento ed invoca il gestore GE opportuno.

- gli eventi che possono essere intercettati sono predefiniti in JAVA (keyPressed, mouseClicked,…..)

JAVA 2D PROGRAMMING: GESTIONE DEGLI EVENTI

Consideriamo Guardie e Ladri: gli eventi di interesse generati dalla *keyBoard* sono:

- pressione delle 4 frecce (up,down,left,right): in questo caso deve essere aggiornata la posizione del ladro controllato dal giocatore, in modo consistente
- pressione della lettera “E”, richiesta al server del livello di energia degli altri giocatori
- pressione tasto ESC per uscita dal gioco

E’ necessario definire un gestore in grado di eseguire le azioni opportune in seguito al verificarsi di questi eventi

JAVA 2D PROGRAMMING: GESTIONE DEGLI EVENTI

Per definire un gestore di eventi è possibile adottare due approcci diversi:

- i*mplementare* tutti i metodi di alcune *interfacce predefinite Listener.* Ad esempio, per gli eventi generati da tastiera l'interfaccia *KeyListener,* che contiene un metodo per ogni tasto che può essere premuto

- estendere alcune classi predefinite, le classi *Adapter.* Ad esempio, nel caso della tastiera, estendere la classe *KeyAdapter.*

Vantaggi del secondo approccio: le classi Adapter implementano le interfacce Listener, per cui forniscono un'implementazione di default per ogni metodo. In questo modo è possibile effettuare l'overriding dei soli metodi che interessano (esempio corrispondenti alla pressione delle 4 frecce,…)

JAVA 2D PROGRAMMING: GESTIONE DEGLI EVENTI

Gestore eventi da KeyBoard:

class keyManager *extends KeyAdapter (implements* Runnable)

public keyManager (…)
{…. }
public keyPressed(*KeyEvent* e)
{if (e.getKeyCode( )==KeyEvent.VK_UP)
{ *System.out.println*("Spostamento in alto");
……}

else if (*e.getKeyCode*( )= =*KeyEvent.VK_DOWN*)
{ *System.out.println*("Spostamento in basso");
……}

JAVA 2D PROGRAMMING: GESTIONE DEGLI EVENTI

public keyPressed(*KeyEvent* e):

- *KeyPressed* nome di un metodo predefinito nella classe *KeyListener.*

  Tale metodo viene invocato quando viene premuto un tasto.

- *KeyEvent* e' un oggetto di tipo *KeyEvent* contiene un codice che descrive il tipo di evento generato

- *e.getKeyCode( )==KeyEvent.VK_UP*

  e.getKeyCode restituisce il codice che specifica quale evento si è verificato in particolare.

  Codici predefiniti

  VK_UP, VK_DOWN,VK_RIGHT,VK_LEFT,….

JAVA 2D PROGRAMMING: GESTIONE DEGLI EVENTI

Come associare un listener ad un frame:

KeyManager km = *new* KeyManager (…);

addKeyListener(km);

Associa il listener ad un'istanza di frame (implicito this.addkeylistener(km))

Analogamente

addMouseListener(..)

addWindowListener()

…..

JAVA 2D PROGRAMMING: GESTIONE DEGLI EVENTI

Devono essere importate le seguenti classi:

```
import java.awt.*;
import java.awt.image.BufferStrategy;
import java.awt.image.BufferedImage;
import java.io;
import java.net.URL;
import javax.imageio.ImageIO;
import java.awt.event.*;
```

JAVA 2D PROGRAMMING: GESTIONE DI DIVERSI FRAME

- nella stessa applicazione è possibile gestire un insieme di frames, corrispondenti a finestre diverse

- Ad esempio: un frame per il rendering, un frame con componenti GUI (bottoni, caselle di testo,…)

- Se non si conosce la GUI JAVA è comunque possibile svolgere tutto il progetto con le nozioni presentate in questi lucidi

ESERCIZIO

Scrivere un'applicazione che consenta all'utente *di muovere* sullo schermo una immagine nelle quattro direzioni *destra, sinistra, alto, basso*. Se l'utente non preme alcun tasto l'immagine continua a spostarsi verso l'ultima direzione indicata dall'utente. L'utente può fermare l'immagine premendo il tasto "S" (stop).

Deve essere anche possibile spostare l'immagine in una posizione qualsiasi dello schermo. Questa posizione viene indicata dall'utente mediante un *click del mouse*.

L'utente termina l'applicazione premendo il tasto ESC.