Programmazione II-A 2020-21

Esercitazione a turni riuniti #3

Attilio Fiandrotti
attilio.fiandrotti@unito.it

29 Aprile 2021

AWT Abstract Windowing Toolkit

- Windowing toolkit grafico originale Java (1995)
- Contiene funzioni per (non esaustivo)
  1. Disegnare primitive grafiche a schermo (*graphical primitives*)
  2. Reagire ad azioni sulla finesta (resize, drag, ...) di una finestra o all'interno della finestra stessa (mouseclick, keystroke, ...) (*event handling*)
  3. Costruire un'interfaccia grafica con elementi annidati gerarchicamente (*toolkit / widgets*)

https://docs.oracle.com/javase/7/docs/api/java/awt/package-summary.html

AWT Abstract Windowing Toolkit

- Disegno di *primitive grafiche* via *java.awt.Graphics*
- Primitive grafiche come metodi
  - *drawLine (int x1, int y1, int x2, int y2)*
  - *drawRect (int x, int y, int width, int height)*
  - *drawString (String str, int x, int y)*
  - *drawOval (int x, int y, int a_x, int a_y);*
  - *setColor (Color col)*
  - *etc.*
- Tipicamente invocate nel metodo *paint(Graphics g)*

java.awt

Class Graphics

java.lang.Object
  java.awt.Graphics

https://docs.oracle.com/javase/7/docs/api/java/awt/Graphics.html

AWT Abstract Windowing Toolkit

- Gestione *eventi* finestre via *callback functions*
- L'interfaccia *WindowListener* consiste in:
  - *windowActivated(WindowEvent e)*
  - *windowDeactivated(WindowEvent e)*
  - *windowOpened(WindowEvent e)*
  - *windowClosing(WindowEvent e)*
  - *windowClosed(WindowEvent e)*
  - *windowIconified(WindowEvent e)*
  - *windowDeiconified(WindowEvent e)*

java.awt.event

Interface WindowListener

All Superinterfaces:

EventListener

https://docs.oracle.com/javase/7/docs/api/java/awt/event/WindowListener.html

AWT Abstract Windowing Toolkit

- Fornisce i *widgets* per costruire GUIs
- Alcune estensioni di java.awt.Component
  - java.awt.Button
  - java.awt.Checkbox
  - java.awt.Label
  - java.awt.Choice

java.awt

Class Component

java.lang.Object
  java.awt.Component

All Implemented Interfaces:

ImageObserver, MenuContainer, Serializable

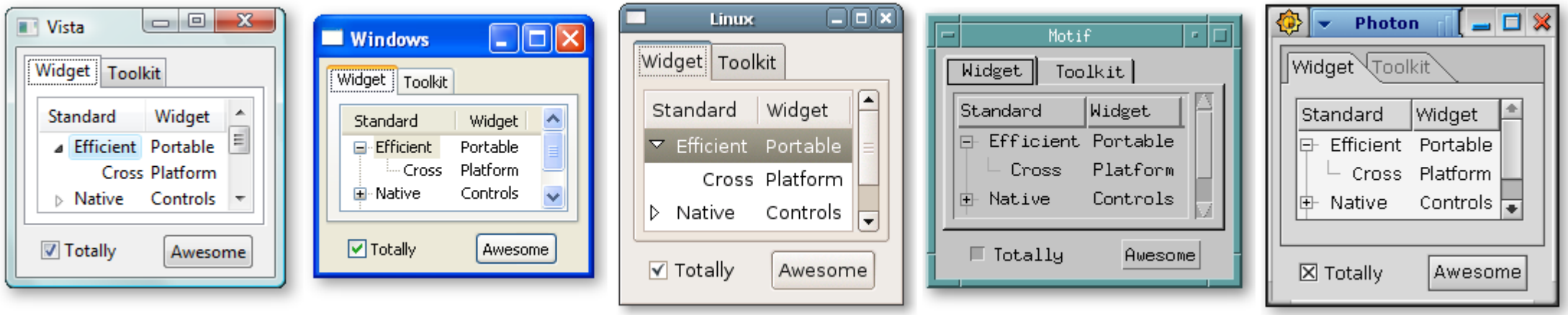
Direct Known Subclasses:

Button, Canvas, Checkbox, Choice, Container, Label, List, Scrollbar, TextComponent

https://docs.oracle.com/javase/7/docs/api/java/awt/Component.html

AWT Abstract Windowing Toolkit

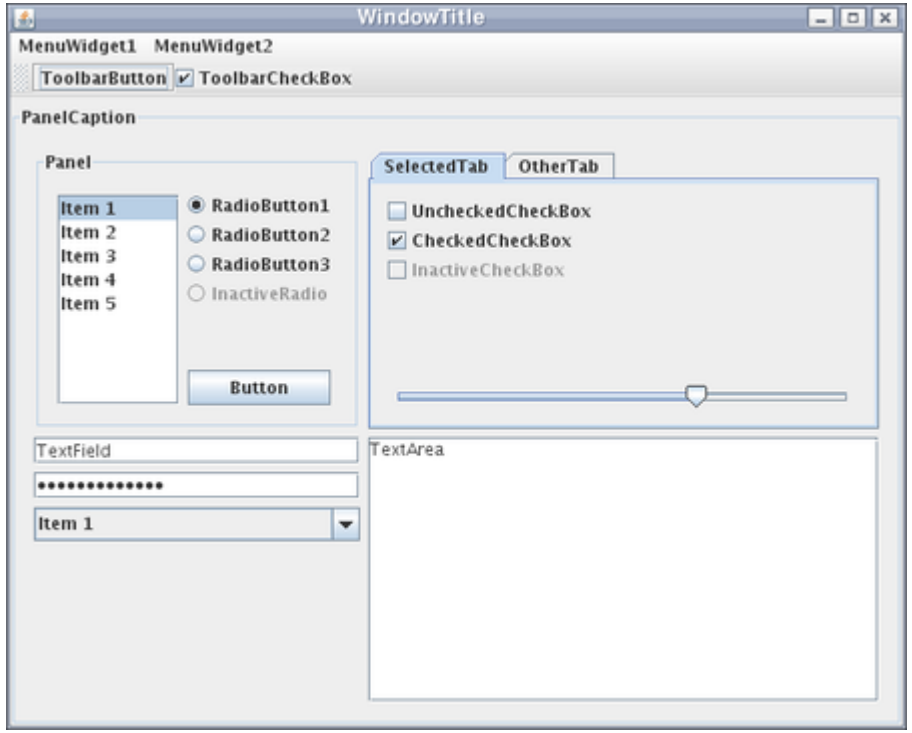
- AWT utilizza il toolkit del sistema host per implementare i widgets
- Lo stesso codice produrrà risultati diversi a seconda del sistema host



https://o7planning.org/10177/which-platform-should-you-choose-for-developing-java-desktop-applications

JAVA Swing

- Introdotto in Java SE 1.2 (1998 circa)
- Implemntazione dei widget nativa in linguaggio JAVA (*lightweight*)
- Stesso risultato su piattaforme diverse (o quasi)
- Altre opzioni oggi possibili per implementare GUIs in Java



https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html

Approccio AWT + Swing

- E’ possibile «combinare» AWT e Swing
- A lezione avete utilizzato *javax.swing.JFrame*
- Estende java.awt.Frame
- E’ una “tavolozza” per elementi grafici
- Il render a schermo è effettuato da
  public void paint(Graphics g)
  esempio: chiamata a *setVisible(True)*
- Potremmo evitare l’uso di Swing ricorrendo a *java.awt.Frame*

javax.swing

Class JFrame

java.lang.Object
  java.awt.Component
    java.awt.Container
      java.awt.Window
        java.awt.Frame
          javax.swing.JFrame

https://docs.oracle.com/javase/7/docs/api/javax/swing/JFrame.html

Lezione 13. Parte 2

- Java contiene la classe ***Graphics*** degli oggetti grafici: un oggetto grafico è un'area rettangolare dello schermo in cui ho dei metodi per disegnare. Vediamo come programmare con la classe Graphics usando l'ereditarietà. Le tappe sono le seguenti.
- A partire da ***Graphics***, definiamo una classe ***Figura*** di figure, ciascuna con un suo metodo ***draw(Graphics g)*** che disegna la figura in un oggetto grafico g. ***Definiamo un metodo draw vuoto per una figura generica***, quindi usando l'ereditarietà ri-definiamo draw ogni volta che definiamo una sottoclasse di Figure.

Lezione 13. Parte 2 (cont)

- A partire da Figura, definiamo la classe ***Disegno*** delle finestre (= oggetti della classe Jframe) con incluso un array di Figure. Noi forniamo un metodo ***void paint(Graphics g)*** che prende in ingresso in un oggetto grafico g posto nella finestra, e disegna tutte le figure dell'array in g usando draw.
- Alla fine, invochiamo su un disegno un metodo di libreria ***setVisible(true)***, che fornisce un oggetto grafico g al metodo paint e genera il disegno. Non ***è possibile definire g*** noi stessi: il costruttore della classe Graphics è ***protected***, dunque ***inaccessibile in Disegno e Figure,*** che non sono sottoclassi di Graphics e non fanno parte della stessa cartella.

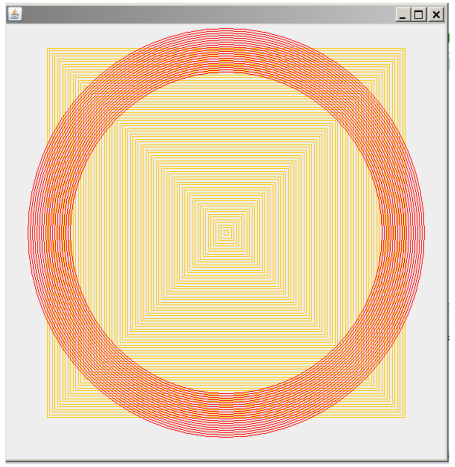
Lezione 13. Parte 2 (cont)

JFrame

Disegno

Figura

```
public void draw(Graphics g){ /* vuoto */ }
```



Quadrato

```
public void draw(Graphics g) {
   g.setColor(Color.orange);
   int m = lato / 2;
   g.drawLine( m,  m, -m,  m);
   g.drawLine(-m,  m, -m, -m);
   g.drawLine(-m, -m,  m, -m);
   g.drawLine( m, -m,  m,  m);
}
```

Cerchio

```
public void draw(Graphics g) {
  g.setColor(Color.red);
  g.drawOval(-raggio,-raggio,  2*raggio,2*raggio);
}
```

```
public class Disegno extends JFrame{
    private Figura[] figure;
    [...]
}
```

```
public void paint(Graphics g) {
    [...]
    //DISEGNO tutte le figure dell'array figure
    for(int i=0;i<figure.length;++i) {
        figure[i].draw(g);
    }
}
```

triggers

```
public static void main(String[] args){
   Figura[] figure = new Figura[n];
   [...]
   Disegno frame = new Disegno(figure)
   [...]
   frame.setVisible(true);
}
```