GUI in Java con l'AWT – 1

Stefano Mizzaro

Dipartimento di matematica e informatica
Università di Udine
http://www.dimi.uniud.it/mizzaro/
mizzaro@uniud.it

Programmazione, lezione 20
12 maggio 2015

Riassunto

- Programmazione strutturata
- OO: TDA, scambio messaggi, eredità, polimorfismo, OO in Java
- Rassegna API
  - `Object, String, Math, System`
  - Eccezioni
  - File
  - Documentazione Javadoc delle API
  - Sorgenti delle API

Oggi

- Continuiamo i "carotaggi"…
- Obiettivi:
  - Darvi conoscenze specifiche (come si lavora su un file)
  - Darvi un metodo (come si cercano le informazioni sulla documentazione delle API)
  - Darvi "agganci" per favorirvi lo studio (più ne sapete più è facile muoversi)

Oggi: GUI in Java, l'AWT

- Esempi, concetti e princîpi alla base
  - Componenti
  - Eventi
    - Gestori eventi (ascoltatori)
    - Classi interne
  - Layout e layout manager (cenni)
- Da fare e da non fare
- Le API dell'AWT

Cos'è una GUI

- Graphical User Interface
  - Interfaccia Utente Grafica
- Interazione utente-programma basata su
  - Linea di comando (shell Unix)
    - Solo testo, tastiera, cursore
  - GUI (Mozilla, MSWord, XEmacs, …)
    - Grafica, finestre, pulsanti, puntatore/mouse, menu
- Finora solo linea di comando…
- Si fanno le GUI in Java? Come?

GUI in Java

- Tutto è un oggetto (istanza)
  - Finestre, pulsanti, … (componenti)
  - Azioni dell'utente! (eventi)
  - Gestori degli eventi
  - Gestori del posizionamento
- AWT (Abstract Window Toolkit): classi predefinite (`java.awt.*`, `java.awt.event.*`, …)
- AWT: GUI indipendenti dalla piattaforma (lo stesso codice funziona su varie piattaforme HW/SW)

Un esempio

- La nostra prima GUI: una finestra vuota che non fa nulla

```
import java.awt.*;
public class GUI1 extends Frame {
  public GUI1 () {
    super();  // (inutile... super("GUI1");)
    setBounds(100,100,300,200); //Dimensioni
    setTitle("GUI1");           //Titolo finestra
    setVisible(true);           //Rende visibile
  }
}
```

- Classe di prova:

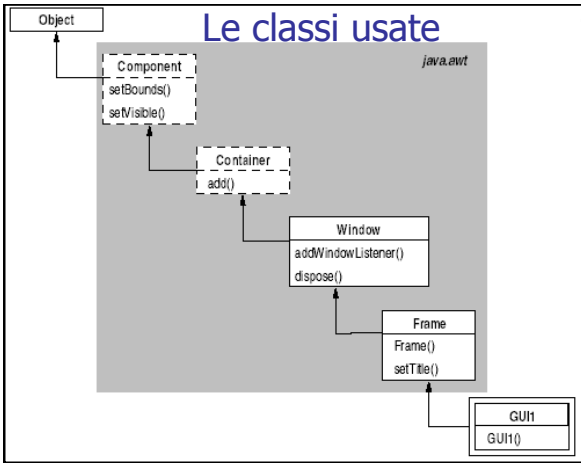
```
public class ProvaGUI1 {
  public static void main (String[] args) {
    GUI1 finestraPrincipale =  new GUI1();
    System.out.println("Fine");
  }
}
```

```
>java ProvaGUI1
```


Come funziona?

- In **`GUI1`** c'è solo il costruttore
- che viene chiamato nel main
- e, usando opportunamente i metodi delle sopraclassi, crea ("costruisce") la finestra
- Notate che "**`Fine`**" viene visualizzato subito…
- Potrei invocare i metodi anche dal main
  - → Ex.
- Cosa c'è in **`Frame`** e nelle superclassi?

Le classi usate



Gestione eventi

- Così com'è **`GUI1`** non fa nulla!
- Ci sono solo componenti, non gestori di eventi
- Modifichiamo la nostra GUI per gestire un evento (chiusura finestra)
- Click su "X" → evento → che codice eseguire?
- Invocazioni implicite: il programmatore
  - scrive il metodo, non la chiamata
  - mette il metodo in un oggetto (ascoltatore)
  - "associa" l'oggetto all'azione

Ascoltatore (1/3)

```
import java.awt.*;
public class GUI2 extends Frame {
  public GUI2 () {
    Ascoltatore asc = new Ascoltatore();
    setBounds(100,100,300,200);
    setTitle("GUI2");
    setVisible(true);
    addWindowListener(asc);//Aggiunge
                           //l'ascoltatore
  }
}
```

```
this.addWindowListener(asc);
```


Ascoltatore (2/3)

```
import java.awt.event.*;
public class Ascoltatore extends WindowAdapter {
  public void windowClosing (WindowEvent e) {
    e.getWindow().dispose();
    System.out.println(e);
  }
  public void windowClosed (WindowEvent e) {
    System.out.println(e);
    System.exit(0);
  }
}
```

- (Ex.: Provare a rimuovere i due metodi uno alla volta…)

Ascoltatore (3/3)

- (quasi nessuna modifica)

```
public class ProvaGUI2 {
  public static void main (String[] args) {
    GUI2 finestraPrincipale = new GUI2();
    System.out.println("Fine");
  }
}
```

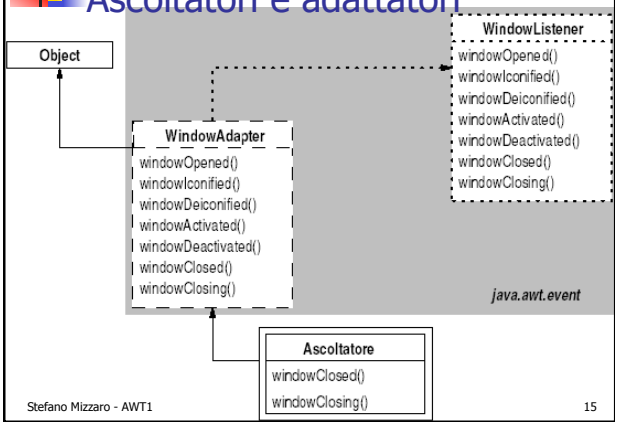
```
>java ProvaGUI2
```

Frame

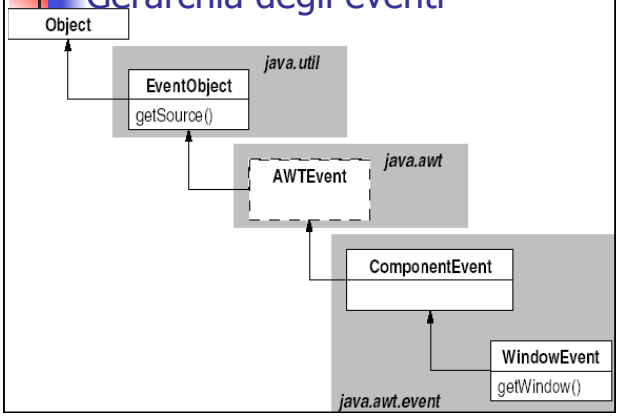
Meccanismo di gestione eventi

- Azione utente →
- Creazione dell'istanza dell'evento opportuno (sottoclasse di **`AWTEvent`**) →
- Invocazione (automatica) dei metodi dell'ascoltatore (un'istanza di **`EventListener`**)
  - L'ascoltatore deve essersi registrato in precedenza per quegli eventi (**`addWindowListener()`**)
  - L'istanza-evento viene passata come argomento al metodo invocato automaticamente
  - È la JVM a invocare un metodo di un'istanza che noi scriviamo

Ascoltatori e adattatori



Gerarchia degli eventi



Tutto è un oggetto!

- 3 istanze:
  - La GUI (**`Frame`**)
  - L'evento generato dal click del mouse sul "pulsante" di chiusura finestra (**`WindowEvent`**)
  - L'ascoltatore con il codice eseguito per gestire l'evento (**`WindowListener`**)
- Facciamo un diagramma unico
- (4a istanza: il gestore della posizione degli oggetti che potrebbero esserci nella finestra)

Senza l'adattatore...

```
import java.awt.event.*;
public class Ascoltatore
                    implements WindowListener {
  public void windowOpened(WindowEvent e) {}
  public void windowIconified(WindowEvent e) {}
  public void windowDeiconified(WindowEvent e) {}
  public void windowActivated(WindowEvent e) {}
  public void windowDeactivated(WindowEvent e) {}
  public void windowClosed (WindowEvent e) {
    System.out.println(e);
    System.exit(0);
  }
  public void windowClosing (WindowEvent e) {
    e.getWindow().dispose();
    System.out.println(e);
  }
}
```

Ascoltatori e classi interne

- Meglio estendere un Adapter che implementare un Listener
- Però c'è eredità singola…
- … e poi è scomodo creare tante classi…
- … e poi come accedo agli attributi privati…
- Soluzione: classi interne per gli ascoltatori
  - Membro ("dentro la classe") (usate questa!)
  - Locali ("dentro un metodo")
  - Anonime ("dentro un metodo e senza nome")

Ascoltatore e classe membro

```
import java.awt.*;
import java.awt.event.*;
public class GUI2 extends Frame {
  class Ascoltatore extends WindowAdapter {
    // idem come prima
  }
  public GUI2 () {
    // idem come prima
  }
}
```


Ascoltatore e classe locale

```
import java.awt.*;
import java.awt.event.*;
public class GUI2 extends Frame {
  public GUI2 () {
    class Ascoltatore extends WindowAdapter {
      // idem come prima
    }
    // idem come prima
  }
}
```


Ascoltatore e classe anonima

```
import java.awt.*;
import java.awt.event.*;
public class GUI2 extends Frame {
  public GUI2 () {
    WindowAdapter asc = (new WindowAdapter() {
      // idem come prima
    });
    // idem come prima
  }
}
```

- o anche…

```
addWindowListener(new WindowAdapter() {
  // idem come prima
});
```


Aggiungiamo qualcosa…

- Ok, ora l'interfaccia fa qualcosa, ma è vuota…
- Aggiungiamo:
  - un'etichetta (c'è la classe `Label`)
  - un pulsante (c'è la classe `Button`)
- `add()`, definito in `Container`

Etichetta e pulsante (1/5)

```
import java.awt.*;
class GUI3 extends Frame {
  public GUI3 () {
    Ascoltatore asc = new Ascoltatore();
    //setBounds(100,100,300,200);
    setTitle("GUI3");
    this.add(new Label("Premi il pulsante!"),
                       BorderLayout.NORTH);
    Button b = new Button("Premimi!");
    Panel p = new Panel();
    p.add(b);
    this.add(p, BorderLayout.SOUTH);
    this.addWindowListener(asc);
    setVisible(true);
  }
}
```

Etichetta e pulsante (2/5)

```
public class ProvaGUI3 {
  public static void main (String[] args) {
    GUI3 finestraPrincipale = new GUI3();
  }
}
```

```
>java ProvaGUI3
```

- Eh, il pulsante non fa nulla…
- … perché manca l'ascoltatore! Aggiungiamolo
  - Click su pulsante → `ActionEvent`
  - Gestito da `ActionListener`

Etichetta e pulsante (3/5)

```
import java.awt.event.*;
class AscoltatorePulsante
                        implements ActionListener {
  public void actionPerformed(ActionEvent e) {
    System.out.println(
                        "Hai premuto il pulsante!");
    System.out.println(e);
  }
}
```

- Note:
  - `ActionListener` non `WindowListener` (né `Action/WindowAdapter`!)
  - `actionPerformed` non `windowClosing`
  - `ActionEvent` non `WindowEvent`

Etichetta e pulsante (4/5)

```
import java.awt.*;
class GUI4 extends Frame {
  public GUI4 () {
    Ascoltatore asc = new Ascoltatore();
    AscoltatorePulsante ap = new
                          AscoltatorePulsante();
    //setBounds(100,100,300,200);
    setTitle("GUI4");
    this.add(new Label("Premi il pulsante!"),
             BorderLayout.NORTH);
    Button b = new Button("Premimi!");
    Panel p = new Panel();
    p.add(b);
    this.add(p, BorderLayout.SOUTH);
    this.addWindowListener(asc);
    b.addActionListener(ap);
    setVisible(true);
  }
}
```

Etichetta e pulsante (5/5)

```
public class ProvaGUI4 {
  public static void main (String[] args) {
    GUI4 finestraPrincipale = new GUI4();
  }
}
```

```
>java ProvaGUI4
```


Commenti

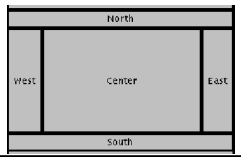
- Pulsante in un Panel
  - Perché?
  - Ex.: provare cosa succede senza il Panel
- Ex.: Fare il diagramma con tutte le istanze
- Ex.: togliere il `setBounds()`
  - che non andrebbe mai usato
  - (e aggiungere `this.pack()`)

Scaletta: AWT

- Esempi, concetti e princîpi alla base
  - Componenti
  - Eventi
    - Gestori eventi (ascoltatori)
    - Classi interne
  - Layout e layout manager (cenni)
- Da fare e da non fare
- Le API dell'AWT

Layout (cenni)

- Come disporre gli oggetti dentro un **`Container`** ("contenitore")
- **`LayoutManager`**: gestore di layout
- Ogni sottoclasse di **`Container`** ha associata un'istanza di **`LayoutManager`**
- Gestore di layout di default (**`BorderLayout`** per **`Frame`**)
- Ne riparleremo…



Da fare e da non fare

- Finestra con due bottoni
- Classe "**`MyFrame`**" che:
  - estende **`Frame`**
  - implementa uno o più listener
- Si può fare in due modi…
  - Male
  - Bene
- Vediamo il codice "male" (tutto in un file)

`MalFatto.java` (1/3)

```
import java.awt.*;
import java.awt.event.*;
public class MalFatto extends Frame
    implements ActionListener, WindowListener{
  Button b1 = new Button("Pulsante 1"),
         b2 = new Button("Pulsante 2");
  public MalFatto() {
    ...
    add(b1); add(b2);
    b1.addActionListener(this);
    b2.addActionListener(this);
    addWindowListener(this);
  }
```


`MalFatto.java` (2/3)

```
public void actionPerformed(ActionEvent e) {
  Object source = e.getSource();
  if (source == b1)
    System.out.println("Premuto Pulsante 1");
  else if (source == b2)
    System.out.println("Premuto Pulsante 2");
  else
    System.out.println("Qualcos'altro...");
}
```


`MalFatto.java` (3/3)

```
  public void windowClosing(WindowEvent e) {
    System.out.println("Window Closing");
    System.exit(0);
  }
  public void windowClosed(WindowEvent e) {}
  public void windowDeiconified(WindowEvent e){}
  public void windowIconified(WindowEvent e) {}
  public void windowActivated(WindowEvent e) {}
  public void windowDeactivated(WindowEvent e){}
  public void windowOpened(WindowEvent e) {}

  public static void main(String[] args) {
    Frame finestra = new MalFatto();
  }
}
```


Perché è fatto male

- Classe che fa 2 cose
  - Costruisce la GUI
  - Gestisce gli eventi
- Tanti metodi vuoti inutili
  - Tanto codice inutile, scritto per niente
- Catene di if/else
- N.B. Purtroppo molti libri usa(va)no questa tecnica…☹
- Vediamo il "bene"

BenFatto.java (1/2)

```
import java.awt.*;
import java.awt.event.*;
public class BenFatto extends Frame {
  public BenFatto() {
    Button b1 = new Button("Pulsante 1"),
           b2 = new Button("Pulsante 2");
    add(b1); add(b2);
    ...
    b1.addActionListener(new
                         AscoltatorePulsante1());
    b2.addActionListener(new
                         AscoltatorePulsante2());
    addWindowListener( new WindowAdapter() {
      public void windowClosing(WindowEvent e){
        System.out.println("Window Closing");
        System.exit(0);
      }
    });
  }
```


BenFatto.java (2/2)

```
  public class AscoltatorePulsante1
               implements ActionListener {
    public void actionPerformed(ActionEvent e){
      System.out.println("Premuto Pulsante 1");
    }
  }
  public class AscoltatorePulsante2
               implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      System.out.println("Premuto Pulsante 2");
    }
  }
  public static void main(String[] args) {
    Frame finestra = new BenFatto();
  }
}
```


Perché è fatto bene

- Evito codice inutile
- Uso variabili locali anziché d'istanza (`b1`, `b2`)
- Ogni classe fa un'unica cosa
  - Porsi sempre la domanda "Che cosa fa questa classe?" per ogni classe che scrivete
  - Se la risposta alla domanda contiene "e", ripensarci…
  - Classe = TDA
  - 1 classe, 1 responsabilità
- Evito catene di if/else, uso l'eredità
- Ascoltatori con classi interne
- Una classe, una responsabilità

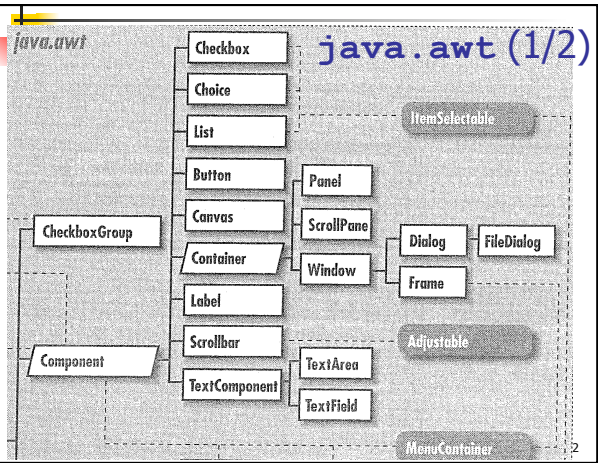
Oggi: GUI in Java, l'AWT

- Esempi, concetti e princîpi alla base
  - Componenti
  - Eventi
    - Gestori eventi (ascoltatori)
    - Classi interne
  - Layout e layout manager (cenni)
- Da fare e da non fare
- Le API dell'AWT

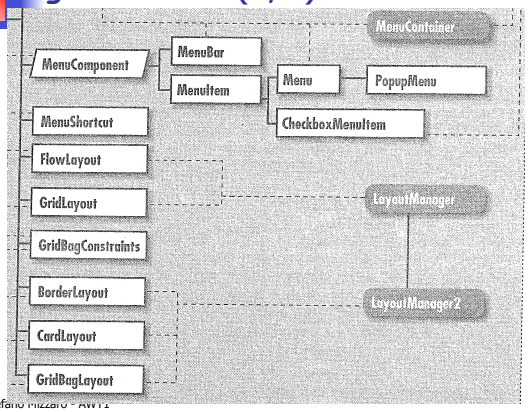
Analisi API: 2 package

- **`java.awt`** (93 classi + 16 interfacce)
  - Contiene le classi per i componenti (finestre, pulsanti, menu, …) e i layout
  - (e altro: **`Font`**, **`Color`**, …)
- **`java.awt.event`** (25 + 18)
  - Contiene le classi per gli eventi
    - Rappresentazione eventi
    - Gestione eventi

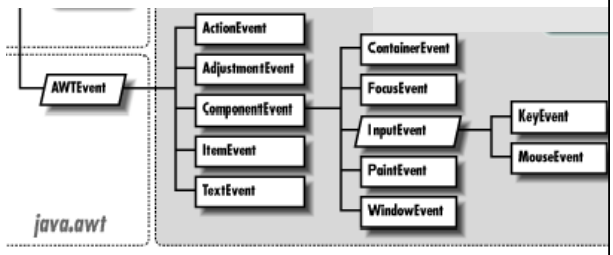
java.awt (1/2)



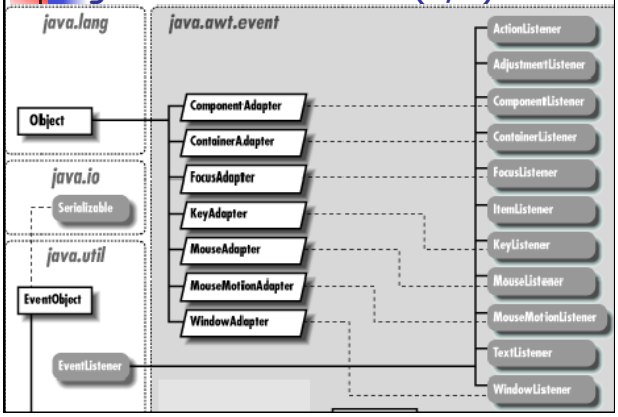
java.awt (2/2)



java.awt.event (1/2)



java.awt.event (2/2)



Riassunto: GUI in Java, l'AWT

- Esempi
- Concetti e princîpi alla base
  - Componenti
  - Eventi
    - Gestori eventi (ascoltatori)
    - Classi interne
  - Layout e layout manager (cenni)
- Da fare e da non fare
- Le API dell'AWT
  - (prox. lezione: analisi più sistematica)

Riferimenti

- Libri & lucidi
- Javadoc API
- Fotocopie
- Tutorial vari in rete