

Upcast - downcast

Conversioni forzate tra tipi riferimento: **casting**

È possibile forzare la conversione da un tipo riferimento **T** ad un sottotipo **T1** purché ...

- ... il tipo **dinamico** dell'oggetto convertito sia un sottotipo di **T1**

Questa conversione **implicita** consentita dal polimorfismo (is-a) viene chiamata **upcast**

```
Object o = new AutomobileElettrica();  
Automobile a = o; // errato, Object non è un  
                // sottotipo di Automobile  
Automobile a = (Automobile) o; // corretto (casting)
```

Questa conversione **esplicita** viene chiamata **downcast**

```
public class Test {  
    public static void main(String a[]) {  
        new Test();  
    }  
}
```

cast

```
Test() {  
    A a;  
    B b = new B();  
    a=b;  
    a.f1();  
    a.f2();  
}  
}
```

OK

NO: "method f2 not found in class
A" (compile time error)

```
class A { void f1()  
    {System.out.println("f1");} }  
class B extends A { void f2()  
    {System.out.println("f2");} }  
class C extends B { void f3()  
    {System.out.println("f3");} }
```

```
public class Test {  
    public static void main(String a[]) {  
        new Test();  
    }  
}
```

cast

```
Test() {  
    A a;  
    B b = new B();  
    a=b;  
    a.f1();  
    ((B) a).f2();  
}  
}
```

upcast implicito

OK: downcast corretto

```
class A { void f1()  
    {System.out.println("f1");} }  
class B extends A { void f2()  
    {System.out.println("f2");} }  
class C extends B { void f3()  
    {System.out.println("f3");} }
```

```
public class Test {  
    public static void main(String a[]) {  
        new Test();  
    }  
}
```

cast

```
Test() {  
    A a;  
    B b = new B();  
    a=b;  
    a.f1();  
    ((C)a).f3();  
}  
}
```

OK: upcast implicit

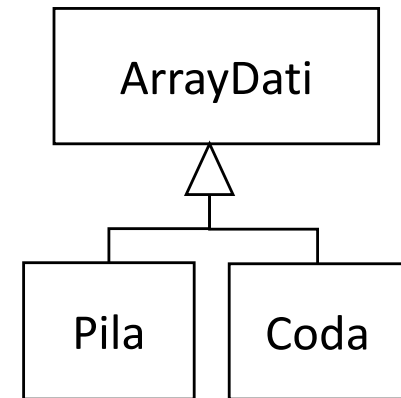
NO: downcast illecito (runtime error)
java.lang.ClassCastException

```
class A { void f1()  
    {System.out.println("f1");} }  
class B extends A { void f2()  
    {System.out.println("f2");} }  
class C extends B { void f3()  
    {System.out.println("f3");} }
```

Ricapitoliamo le vicende del
binding...

Decisioni al volo...

```
public static void main(String  
a[]){  
    ArrayDati p;  
    // leggi k  
    if (k==1) p = new Pila();  
    else p = new Coda();  
    p.inserisci(1);  
    p.inserisci(2);  
    p.estrai();  
}
```

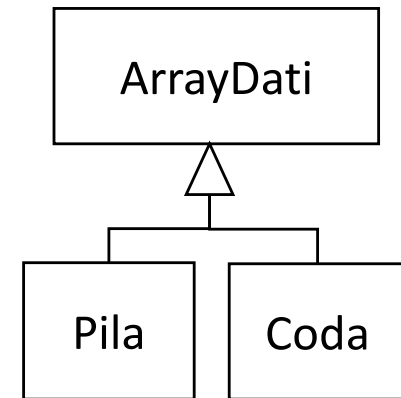


Il tipo di **p** viene deciso a **runtime**!

Il legame tra un oggetto e il suo tipo è **dinamico**
(*dynamic binding*, *late binding*, o *lazy evaluation*)

Domanda: perché non fare così?

```
public static void main(String a[]){  
    //ArrayDati p;  
    // leggi k  
    if (k==1) { Pila p = new Pila();  
    } else { Coda p = new Coda();  
    }  
    p.inserisci(1);  
    p.inserisci(2);  
    p.estrai();  
}
```

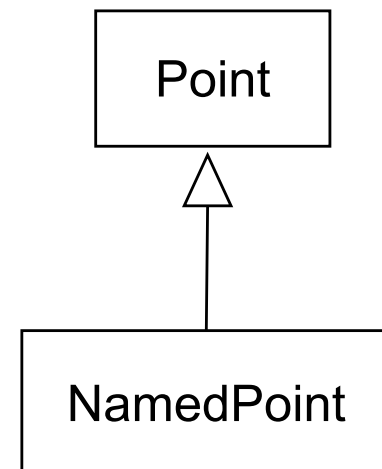


Determinazione del tipo: **instanceof**

È possibile determinare il tipo dinamico di un oggetto con l'operatore **instanceof**

Utile per evitare errori di tipo a runtime dovuti a downcast

```
public static void main(String a[]){  
    Point p;  
    // leggi k  
    if (k==1) p = new Point(2,2);  
    else p = new NamedPoint(3,3,"A");  
    p.getName();  
    if (p instanceof NamedPoint)  
        ((NamedPoint) p).getName();  
}
```



Interface

Interfacce

- Un'interfaccia è una classe Completamente astratta, senza attributi (solo una collezione di firme di metodi pubblici e astratti)

- Sintassi:

```
-interface <nome> {  
-  <lista metodi: solo firme, senza corpo>  
-}
```

- Un'interfaccia può contenere costanti.

- Talvolta si usano interfacce completamente vuote (senza metodi) per «etichettare» classi con speciali proprietà (*tagging interfaces*)

- Es. **Cloneable**, **Serializable**, **Remote**, ...

Interfacce ed ereditarietà

- Una interfaccia può ereditare da **una o più** interfacce (ma non da classi!)

- **interface <nome> extends
<nome₁>, . . . , <nome_n> { . . . }**

Interfacce ed ereditarietà

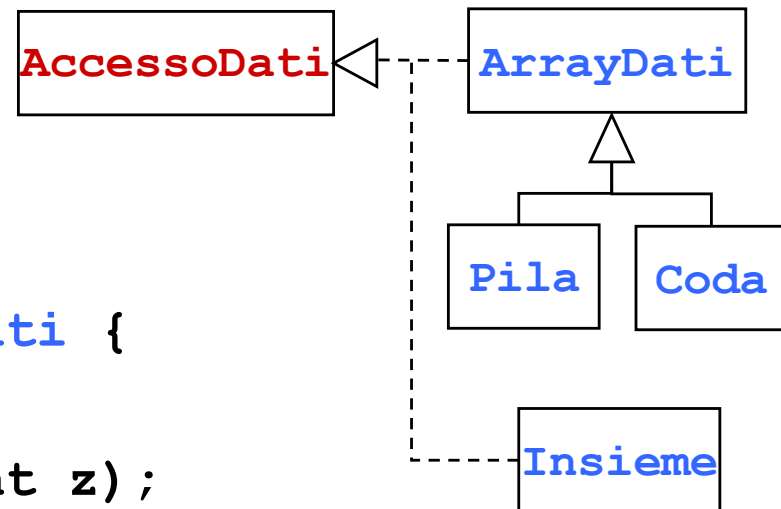
- Una classe può implementare **una o più** interfacce, e DEVE implementarne tutti i metodi
- (a meno che non sia astratta)

```
• class <nome> implements  
  <nome1>, . . . , <nomen> { . . . }
```

Una classe definisce che un oggetto **è** qualcosa; un'interfaccia rappresenta i servizi (**comportamento**) che la classe deve fornire

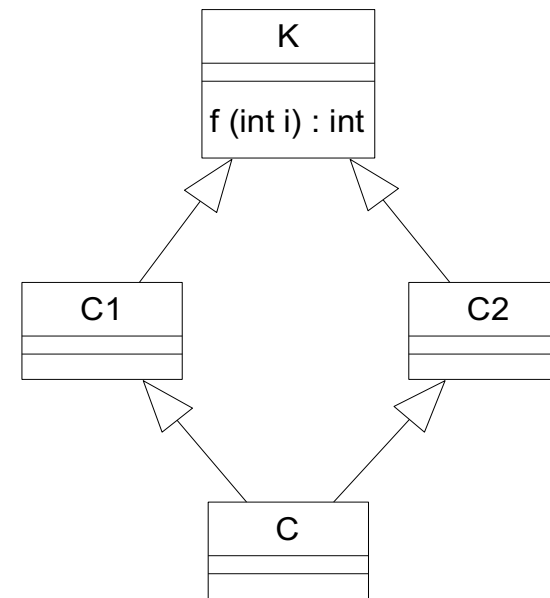
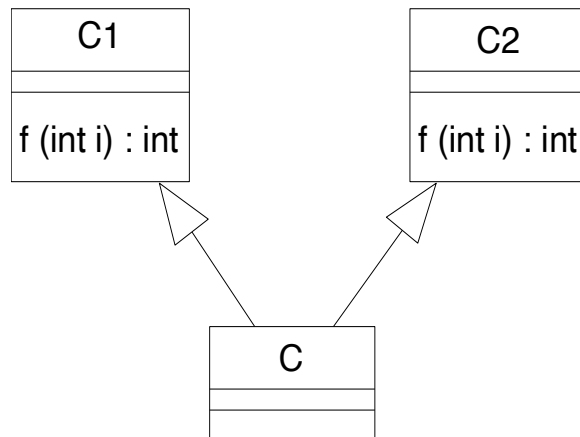
- package strutture;
- public interface AccessoDati {
 - public int estrai();
 - public void inserisci(int z);
- }
- public abstract class ArrayDati implements AccessoDati { ... }
- public class Pila extends ArrayDati { ... }
- public class Coda extends ArrayDati { ... }
- public class Insieme implements AccessoDati {...}

Esempio

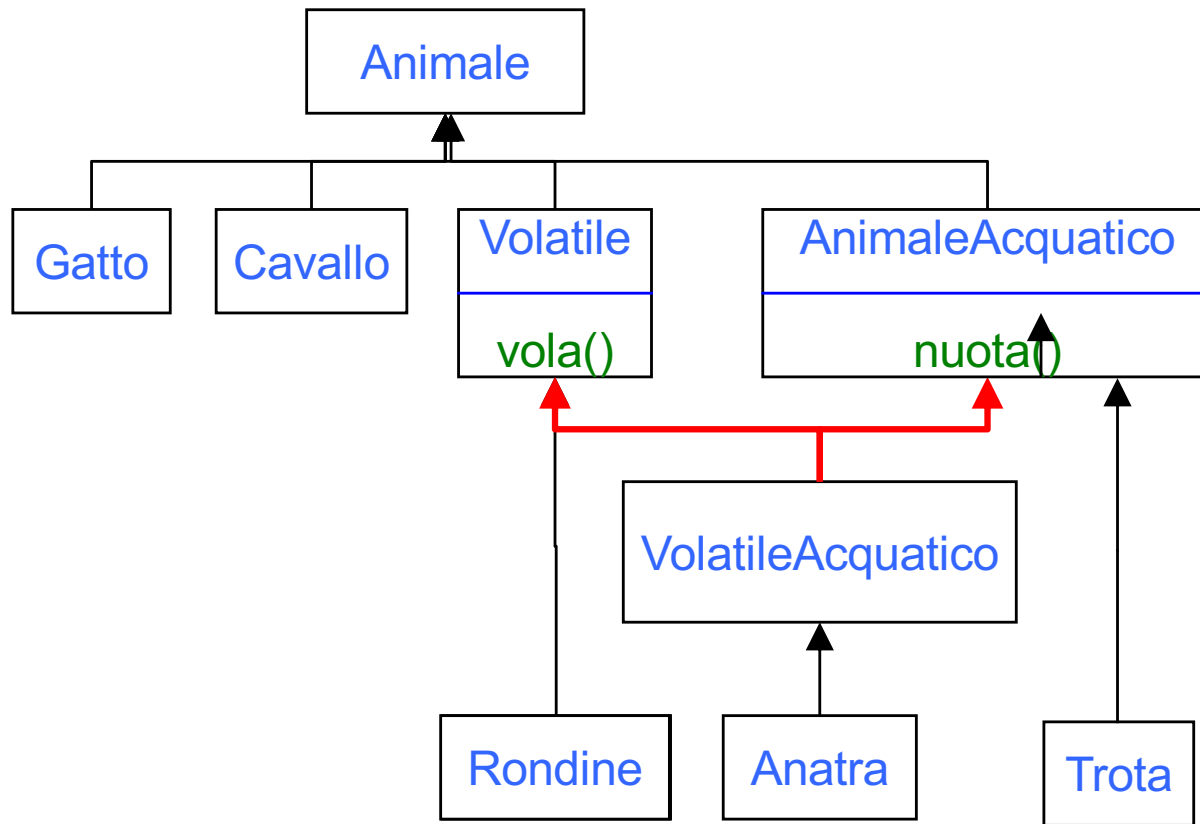


interface resolve i problemi della multiple inheritance

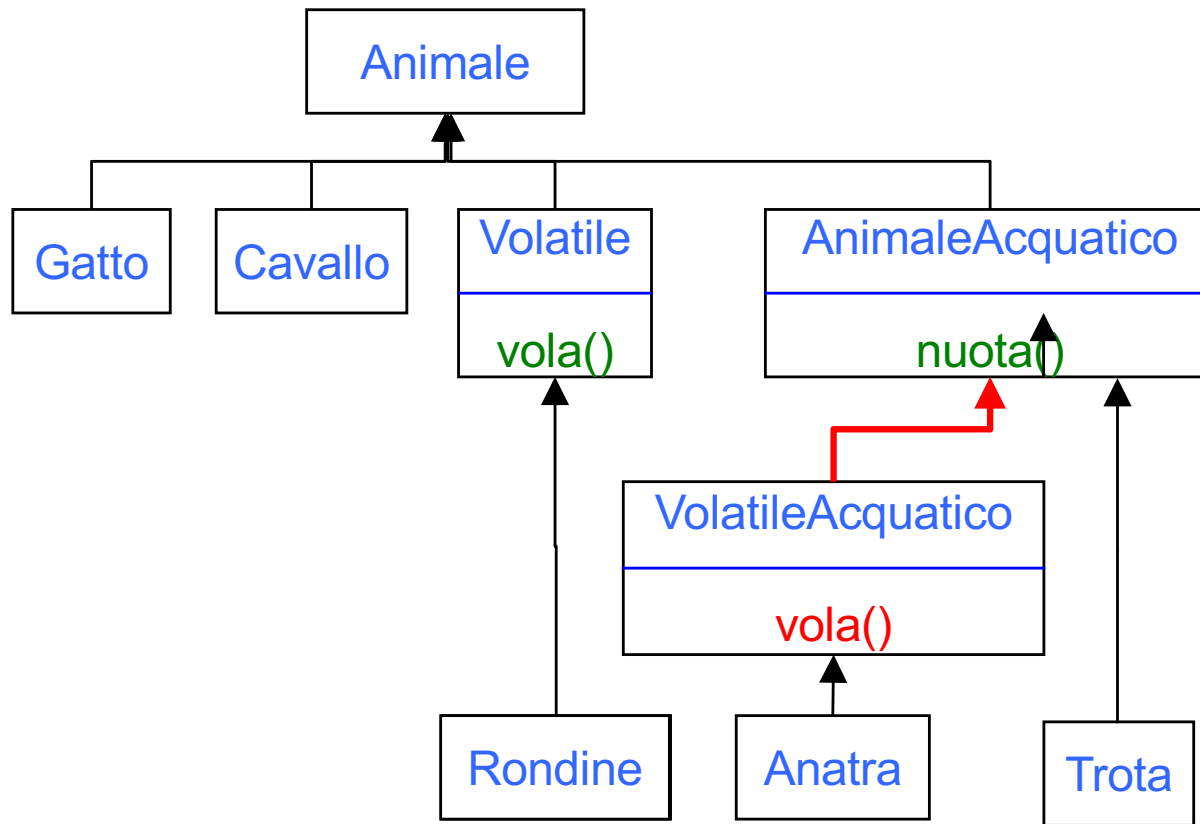
- Nei linguaggi che supportano ereditarietà multipla (es., C++) è possibile ereditare due o più metodi con la stessa firma da più superclassi...
- ... il che crea un conflitto tra **implementazioni** diverse



Esempio: senza interfacce, eredità multipla

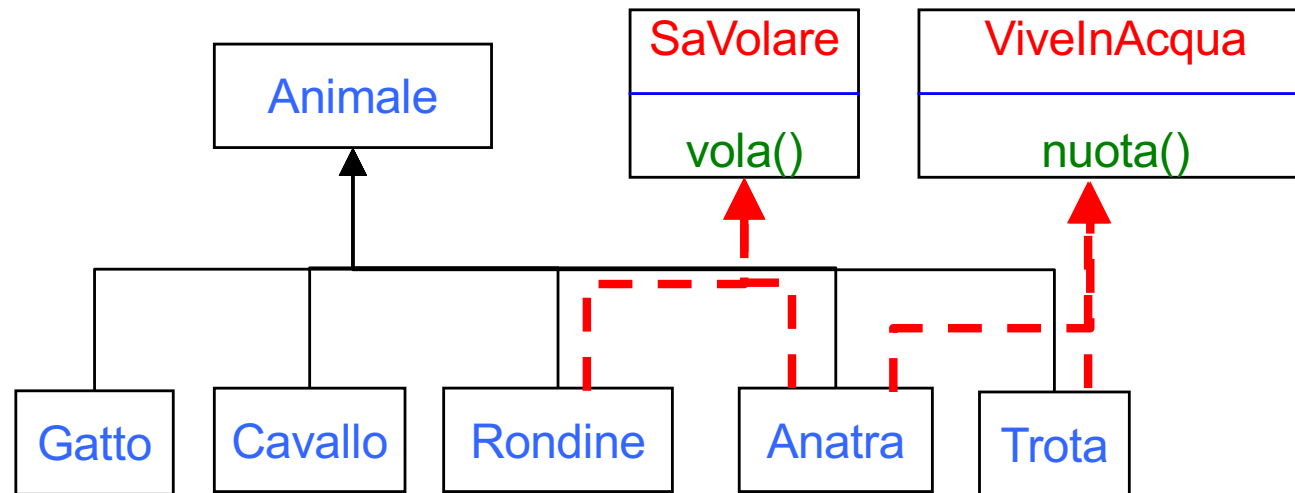


Esempio: senza interfacce, eredità singola



Approccio tassonomico

Esempio: con le interfacce e multiple inheritance



“la rondine è un animale che sa volare”

“l’anatra è un animale che sa volare e vive in acqua”

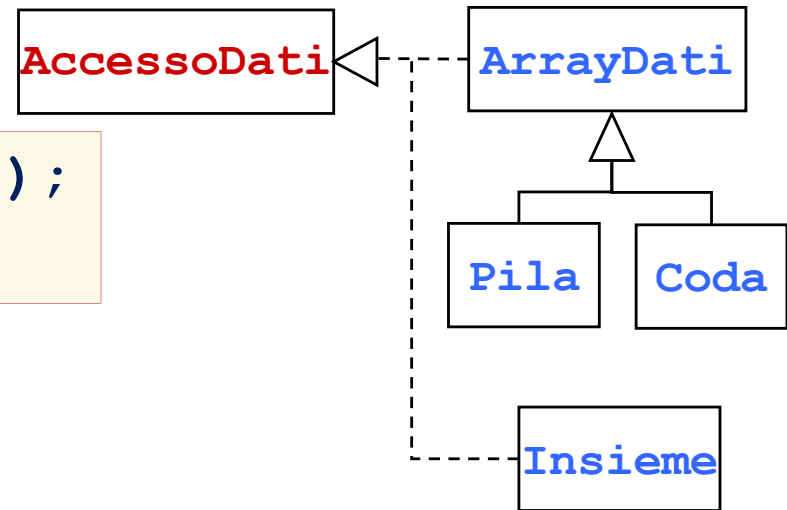
Polimorfismo ed interfacce

- Una interfaccia può essere utilizzata per definire il tipo di una variabile

```
AccessoDati o = new Pila();  
o.inserisci(5);
```

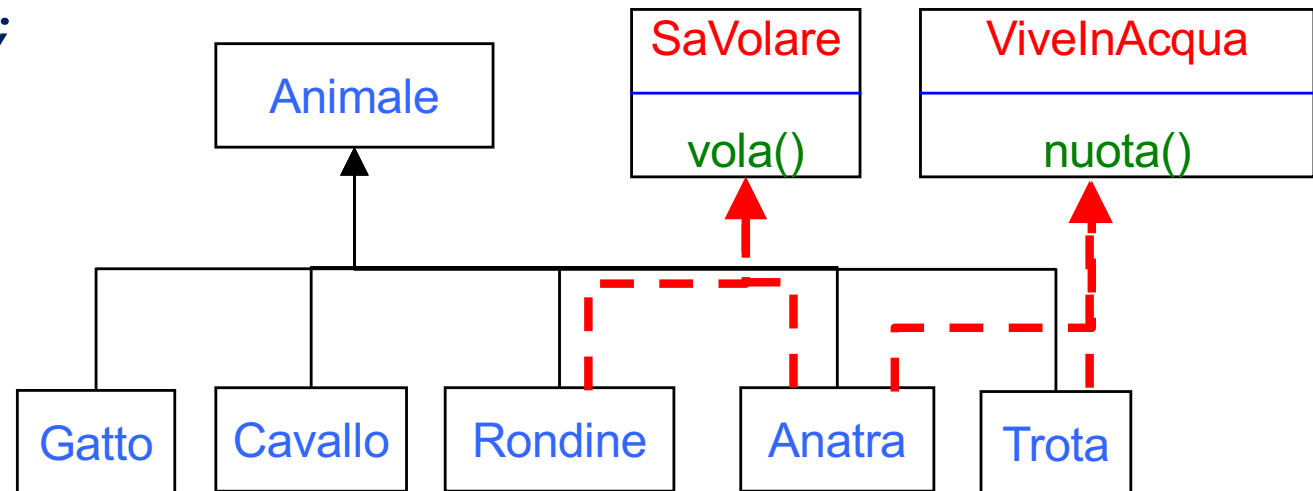
Ma un'interfaccia non
può essere usata per
creare un oggetto!

```
AccessoDati o = new AccessoDati();
```



Esercizio: quali di queste sono errate?

- `Animale g = new Gatto();`
- `Acquatico t = new Trota();`
- `Anatra a = new Anatra();`
- `Acquatico c = new Acquatico();`
- `Volatile l = g;`
- `Volatile v = a;`
- `Acquatico q = a;`
- `g.vola();`
- `v.vola();`
- `t.nuota();`
- `t.vola();`
- `a.nuota();`



Soluzione

- `Animale g = new Gatto();` *// ok*
- `Acquatico t = new Trota();` *// ok*
- `Anatra a = new Anatra();` *// ok*
- `Acquatico c = new Acquatico();` *// errato*
- `Volatile l = g;` *// errato*
- `Volatile v = a;` *// ok*
- `Acquatico q = a;` *// ok*
- `g.vola();` *// errato*
- `v.vola();` *// ok*
- `t.nuota();` *// ok*
- `t.vola();` *// errato*
- `a.nuota();` *// ok*

Conversione di stringhe in numeri

Conversione di stringhe in numeri

```
String s="10";  
int i=Integer.parseInt(s);
```

```
String pi="3.1415026535";  
float  $\pi$ =Float.parseFloat(pi);
```

Che succede se faccio

```
String s="pippo";  
int i=Integer.parseInt(s);
```



Errore!

Exception in Application start method

java.lang.reflect.InvocationTargetException

```
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:498)
    at com.sun.javafx.application.LauncherImpl.launchApplicationWithArgs(LauncherImpl.java:389)
    at com.sun.javafx.application.LauncherImpl.launchApplication(LauncherImpl.java:328)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:498)
    at sun.launcher.LauncherHelper$FXHelper.main(LauncherHelper.java:767)
```

Caused by: java.lang.RuntimeException: Exception in Application start method

```
    at com.sun.javafx.application.LauncherImpl.launchApplication1(LauncherImpl.java:917)
    at com.sun.javafx.application.LauncherImpl.lambda$launchApplication$155(LauncherImpl.java:182)
    at java.lang.Thread.run(Thread.java:745)
```

Caused by: java.lang.NumberFormatException: For input string: "z"

```
    at sun.misc.FloatingDecimal.readJavaFormatString(FloatingDecimal.java:2043)
    at sun.misc.FloatingDecimal.parseFloat(FloatingDecimal.java:122)
    at java.lang.Float.parseFloat(Float.java:451)
    at javafxapplication27.JavaFXApplication27.start(JavaFXApplication27.java:36)
    at com.sun.javafx.application.LauncherImpl.lambda$launchApplication1$162(LauncherImpl.java:863)
    at com.sun.javafx.application.PlatformImpl.lambda$runAndWait$175(PlatformImpl.java:326)
    at com.sun.javafx.application.PlatformImpl.lambda$null$173(PlatformImpl.java:295)
    at java.security.AccessController.doPrivileged(Native Method)
    at com.sun.javafx.application.PlatformImpl.lambda$runLater$174(PlatformImpl.java:294)
    at com.sun.glass.ui.InvokeLaterDispatcher$Future.run(InvokeLaterDispatcher.java:95)
```

Exception running application javafxapplication27.JavaFXApplication27

Gestione degli errori

Proteggiamo questo codice

```
Scanner scanner = new Scanner(System.in);  
String inputString;  
int z;  
System.out.println("dammi un numero");  
inputString= scanner.nextLine();  
z=Integer.parseInt(inputString);  
System.out.println("input valido!");
```

Blocco try - catch

```
Scanner scanner = new Scanner(System.in);
String inputString;
int z;
System.out.println("dammi un numero");
inputString= scanner.nextLine();
try {
    int z=Integer.parseInt(inputString);
    System.out.println("input valido!");
} catch (NumberFormatException ex) {
    System.out.println("input non valido!");
}
```

Possiamo fare di meglio

```
Scanner scanner = new Scanner(System.in);
String inputString;
int z;
boolean failure=true;
do {
    try {
        System.out.println("dammi un numero");
        inputString= scanner.nextLine();
        int z=Integer.parseInt(inputString);
        failure=false;
    } catch (NumberFormatException ex) {
        failure=true;
    }
} while (failure);
```

Clausola finally

Nota: c'è una ulteriore clausola!

```
try {  
    codice che potrebbe generare errore  
} catch (NumberFormatException ex) {  
    codice da eseguire se si verifica un errore  
} finally {  
    codice da eseguire sia che ci sia stato un errore,  
    sia che non ci sia stato.  
}
```

finally

The finally block **always executes** when the try block exits. This ensures that the finally block is executed **even if an unexpected exception occurs**. But finally is useful for more than just exception handling — it allows the programmer to avoid having **cleanup code** accidentally bypassed by a return, continue, or break. **Putting cleanup code in a finally block is always a good practice, even when no exceptions are anticipated.**

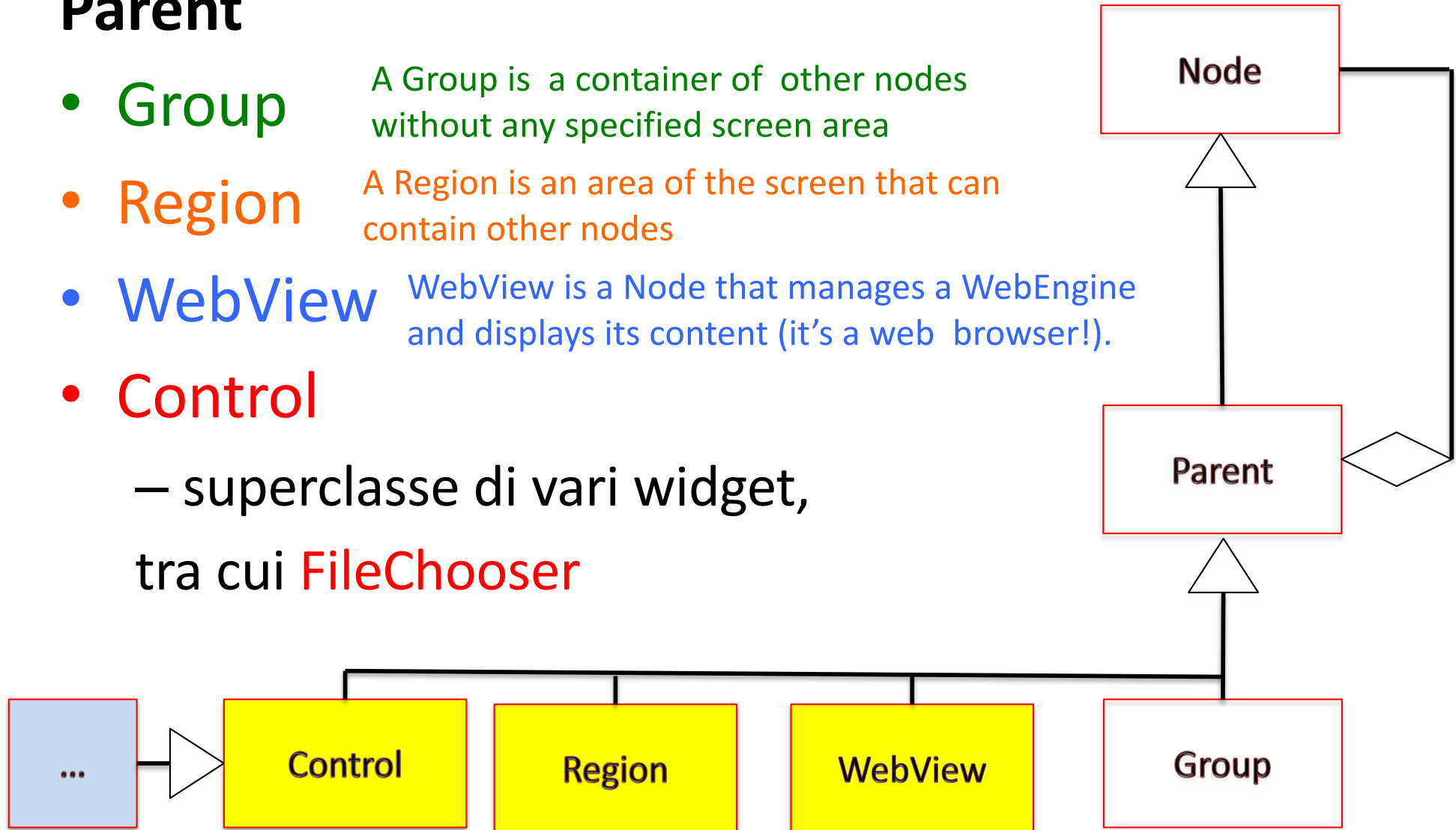
Palestra di Java con la grafica:

Java FX - parte 2

Parent hierarchy

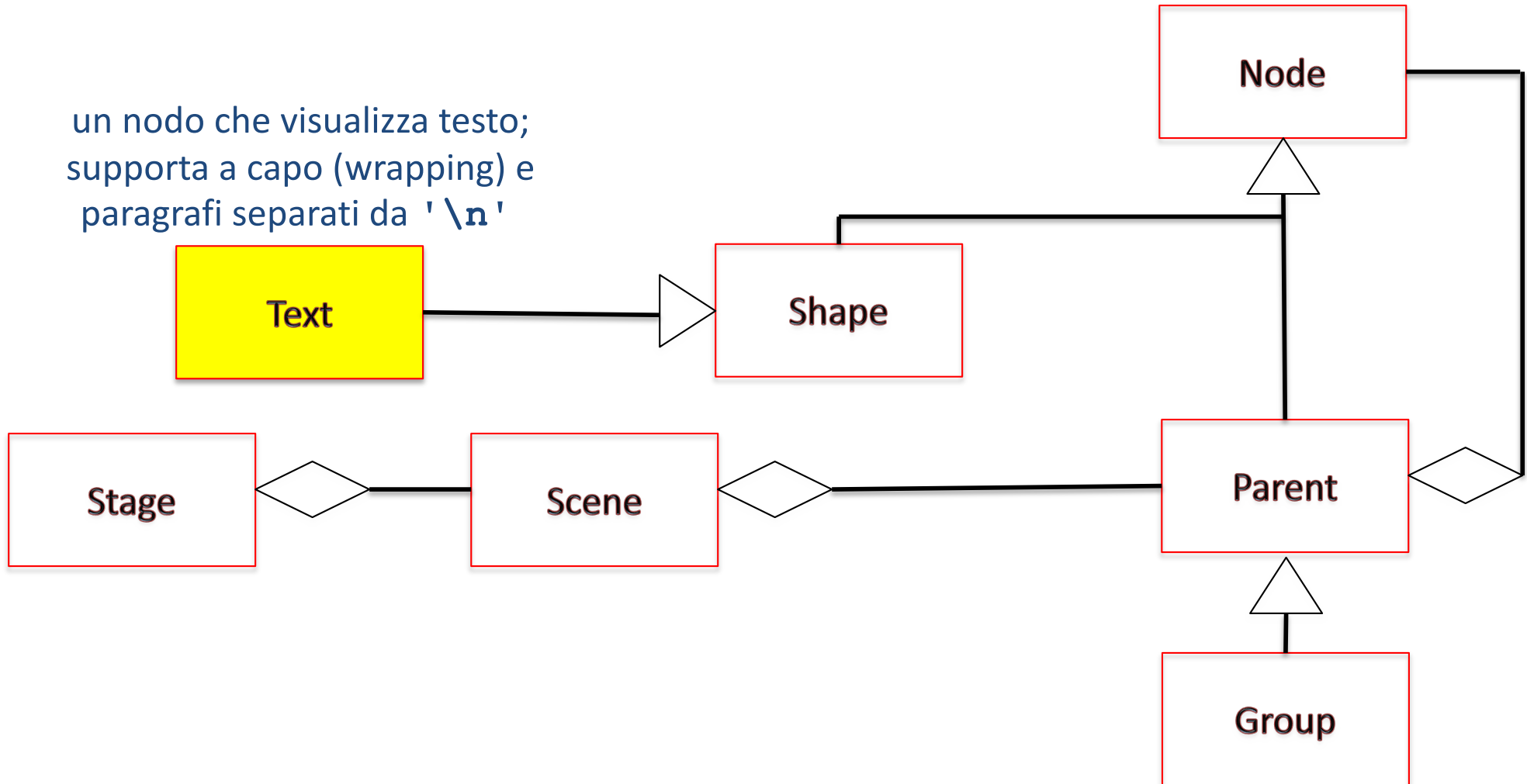
Parent

- **Group** A Group is a container of other nodes without any specified screen area
- **Region** A Region is an area of the screen that can contain other nodes
- **WebView** WebView is a Node that manages a WebEngine and displays its content (it's a web browser!).
- **Control**
 - superclasse di vari widget, tra cui **FileChooser**



Shape & Text

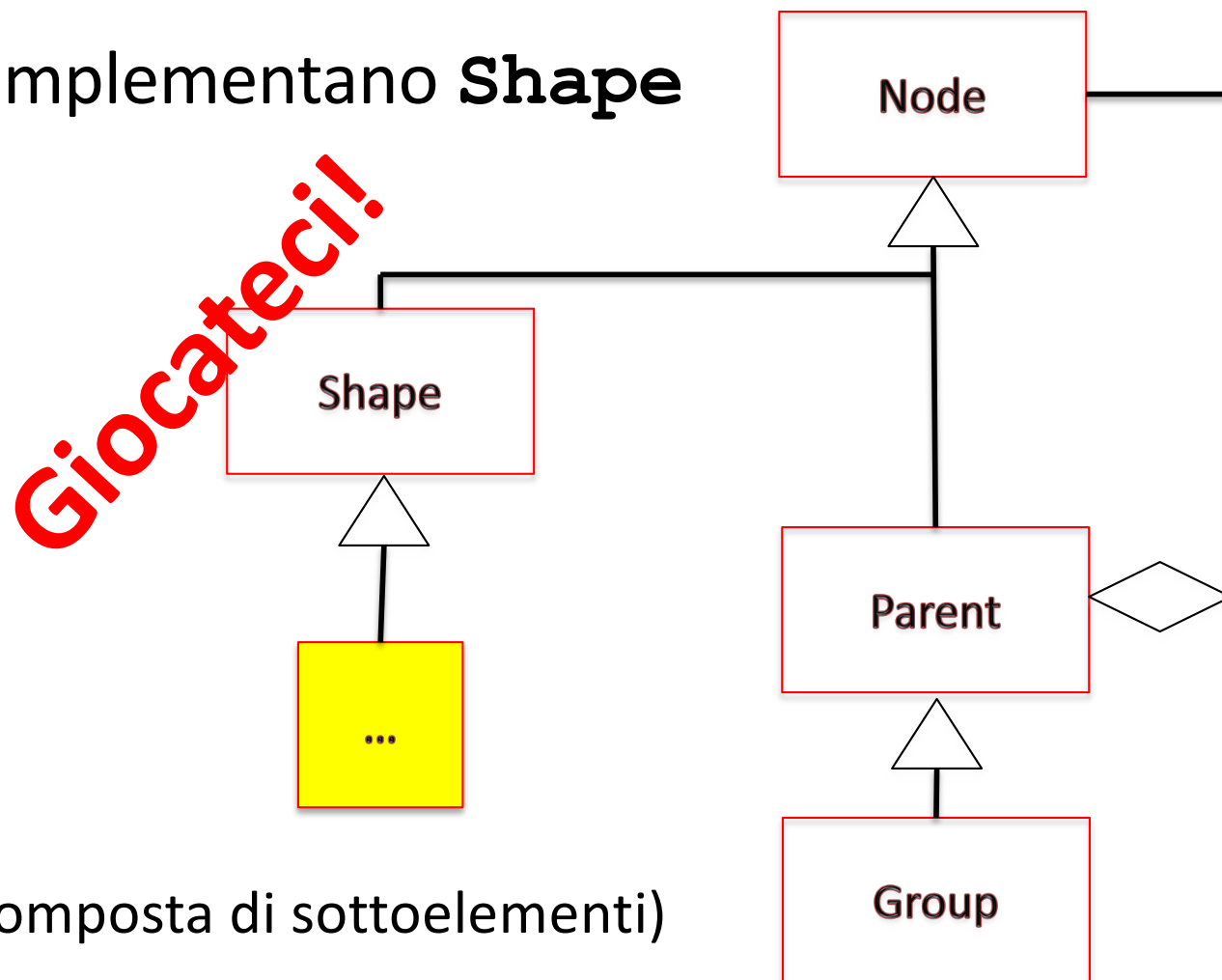
un nodo che visualizza testo;
supporta a capo (wrapping) e
paragrafi separati da '`\n`'



Gerarchia di Shape

Le seguenti classi implementano **Shape**

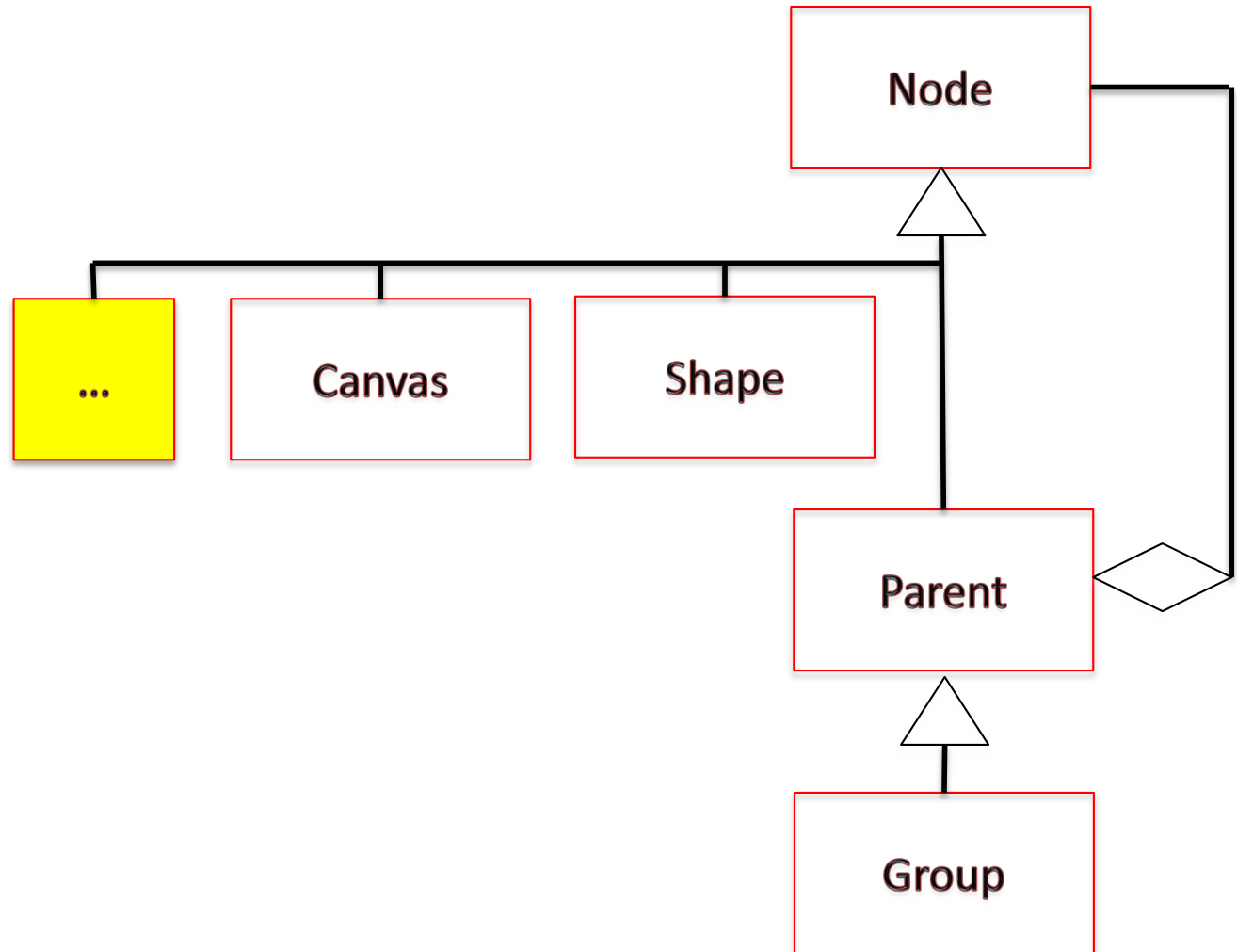
- Line
- Polyline
- Polygon
- Rectangle
- Arc
- Circle
- Ellipse
- QuadCurve
- CubicCurve
- Text
- SVGPath (linea composta di sottoelementi)



Node hierarchy

Node

- Parent
- Shape
- Canvas



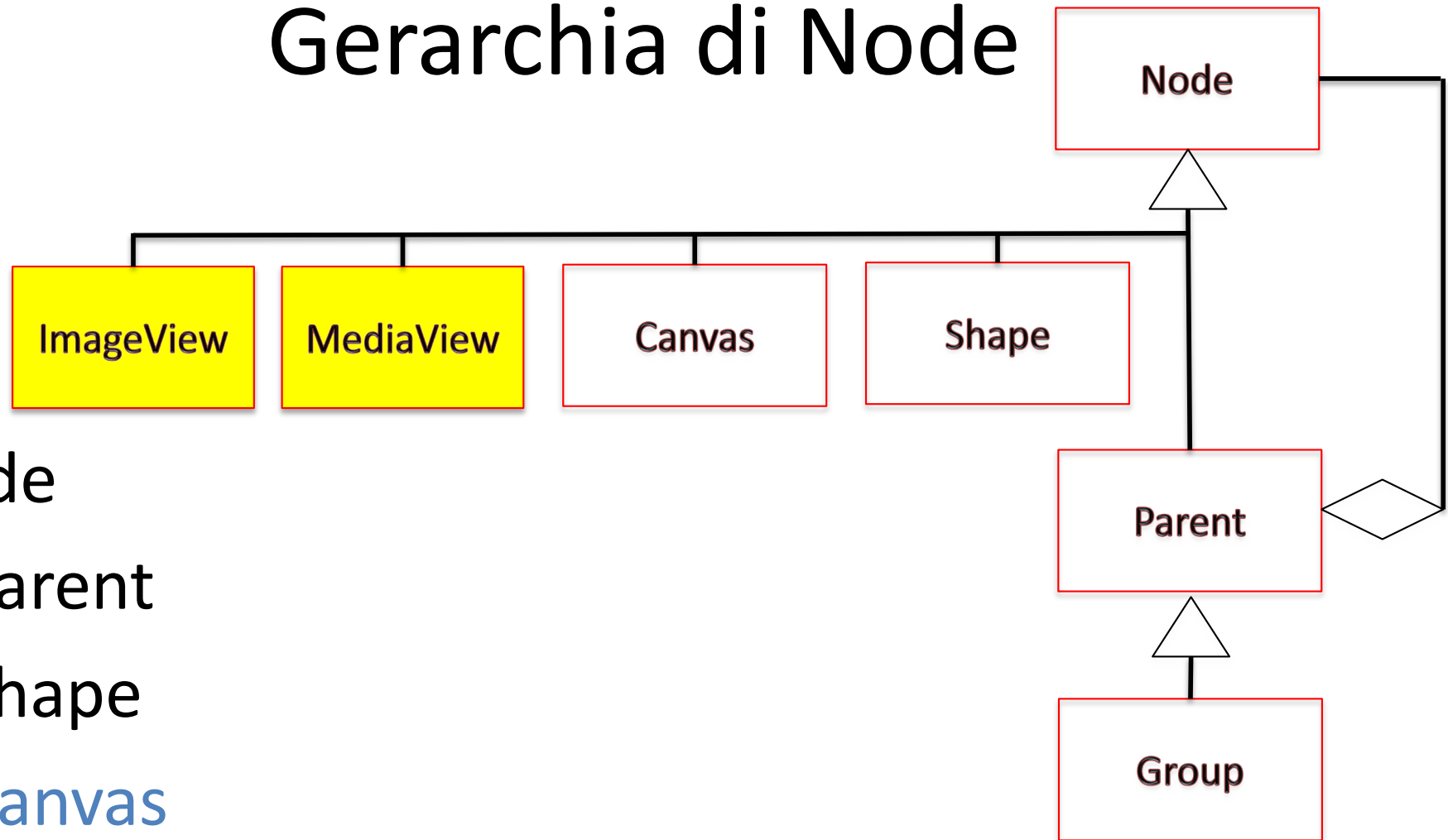
Canvas

Una "tela del pittore" con un metodo per ottenere il suo **GraphicsContext** con varie primitive per disegnarci sopra:

- **fillArc()**
- **fillRect()**
- **drawImage()**
- ...

<http://docs.oracle.com/javafx/2/canvas/jfxpub-canvas.htm>

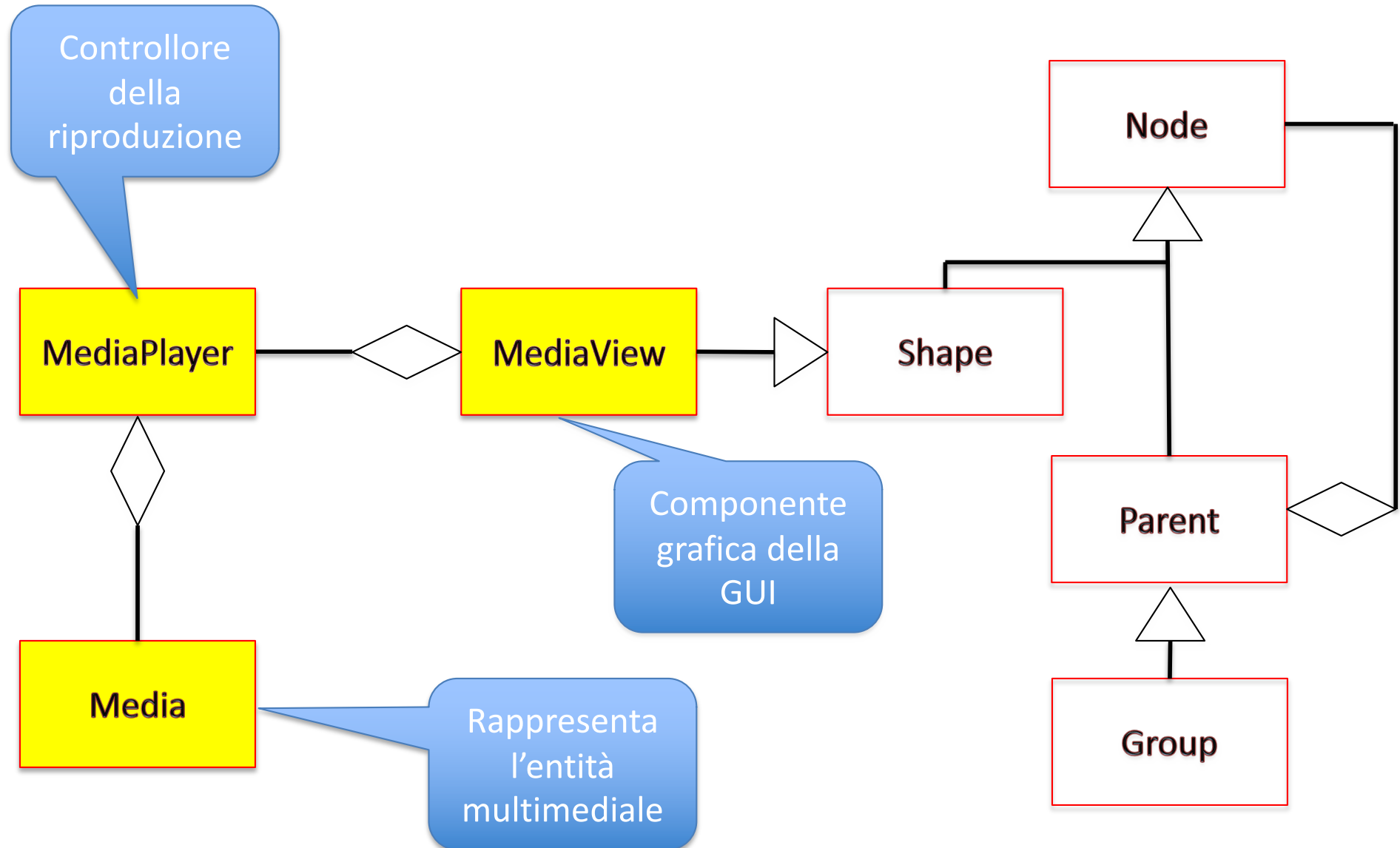
Gerarchia di Node



Node

- Parent
- Shape
- Canvas
- ImageView
- MediaView

MediaView & Media



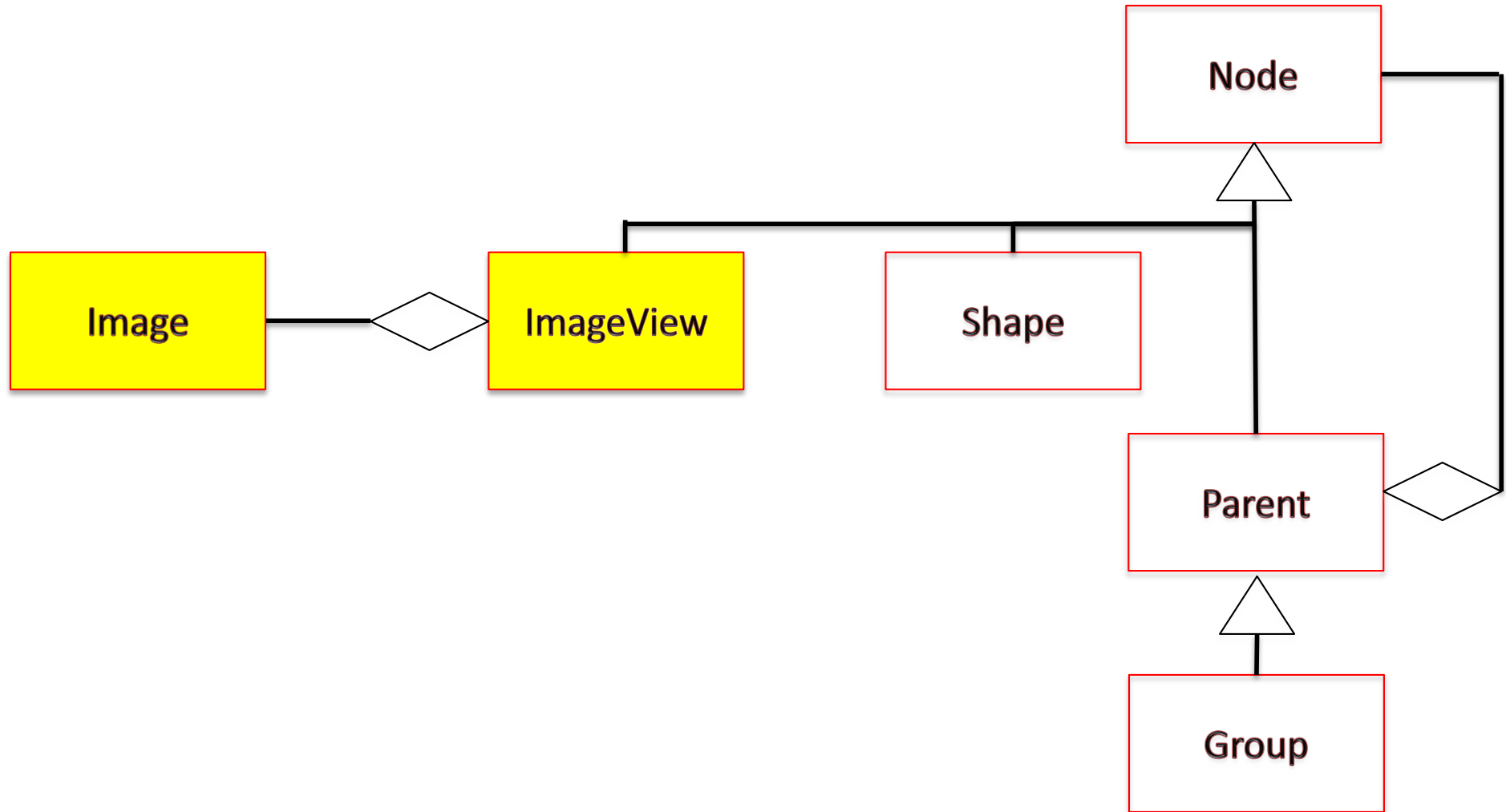
MediaView

```
public class Sounds extends Application{  
    public void start(Stage stage) {  
        Media media = new  
Media("http://www.ferraraterraeacqua.it/it/audioguide/audioguide-  
di-ferrara-citta-del-rinascimento/01_benvenuto-a-ferrara.mp3");  
        MediaPlayer mediaPlayer = new MediaPlayer(media);  
        mediaPlayer.setAutoPlay(true);  
        // create mediaView and add media player to the viewer  
        MediaView mediaView = new MediaView(mediaPlayer);  
        Group root = new Group(mediaView);  
        root.getChildren().add(  
            new Text(10, 30, "Benvenuto a Ferrara"));  
        Scene scene = new Scene(root, 150, 60);  
        stage.setScene(scene);  
        stage.sizeToScene();  
        stage.show();  
    }  
    public static void main(String[] args) {  
        Application.launch(args);  
    }  
}
```

<http://docs.oracle.com/javafx/2/media/overview.htm>
<https://docs.oracle.com/javase/8/javafx/media-tutorial/overview.htm>



ImageView & Image



Gerarchia di Parent (parziale)

Parent

- **Control**
 - superclasse di vari widget, tra cui **FileChooser**

• ...

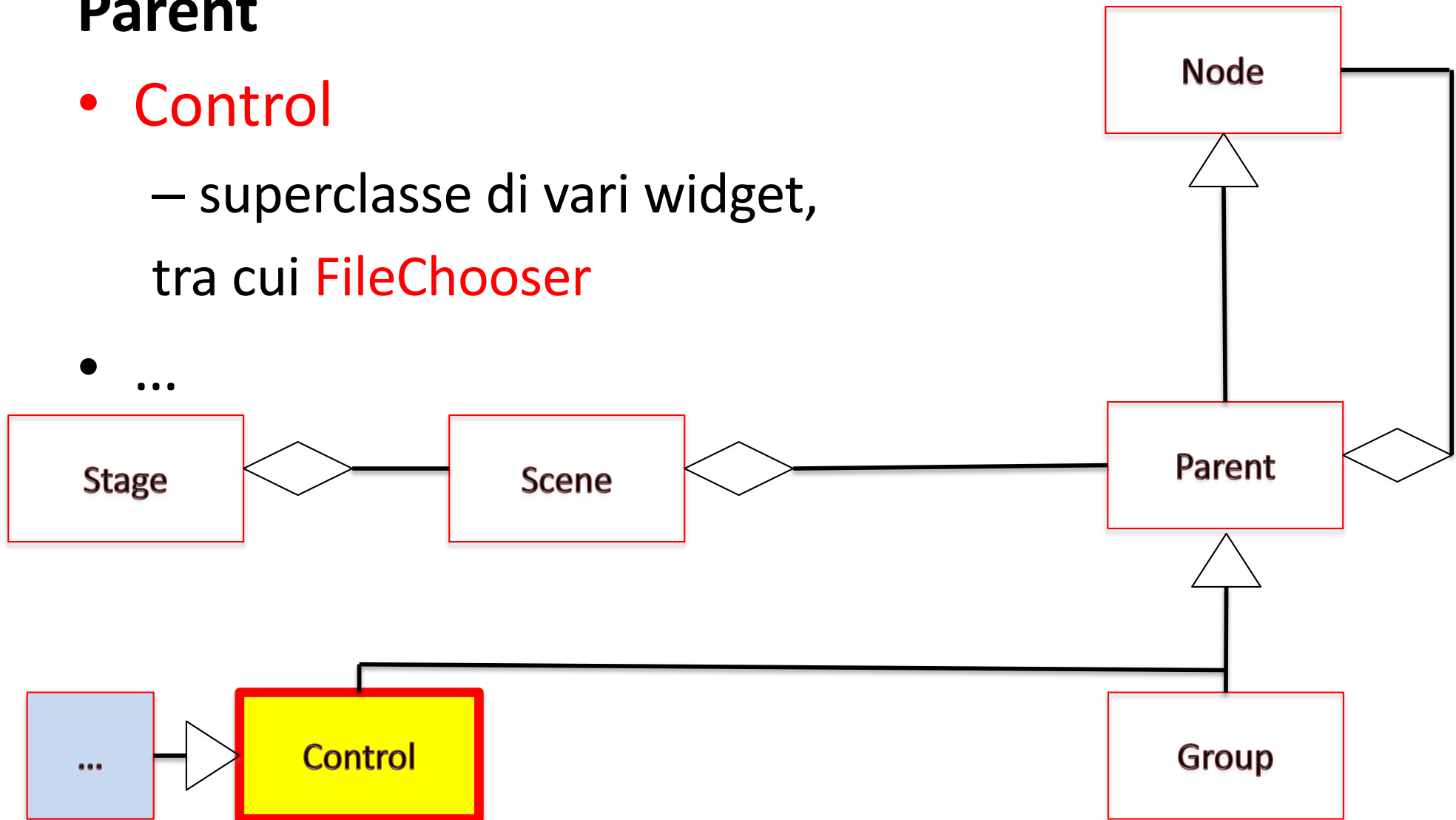
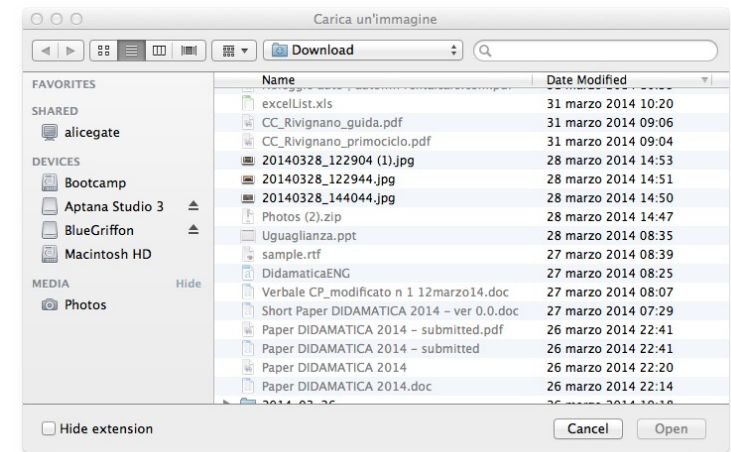


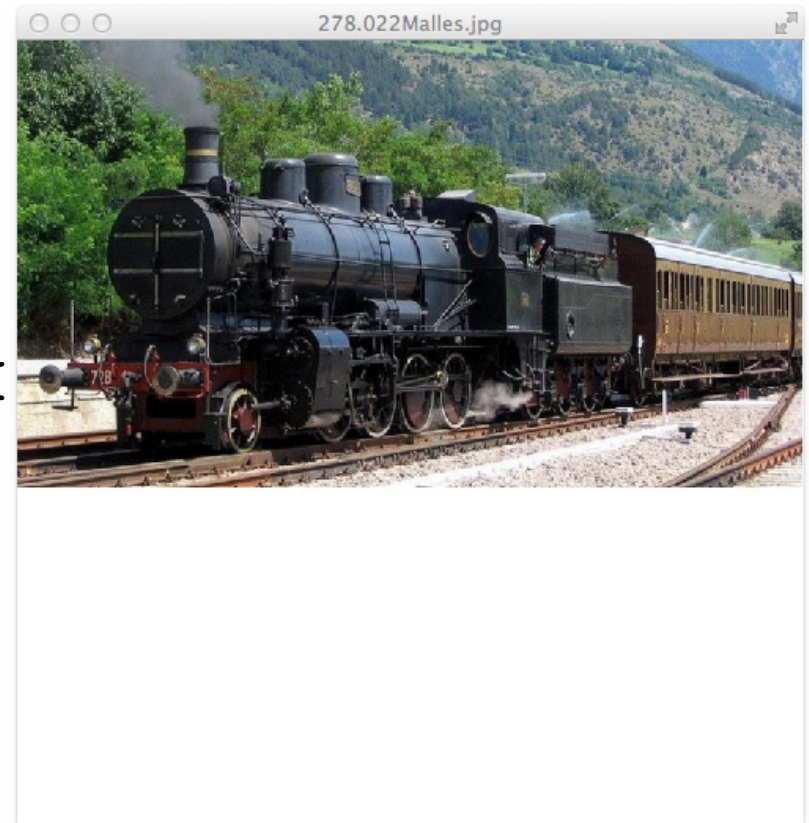
Image & File



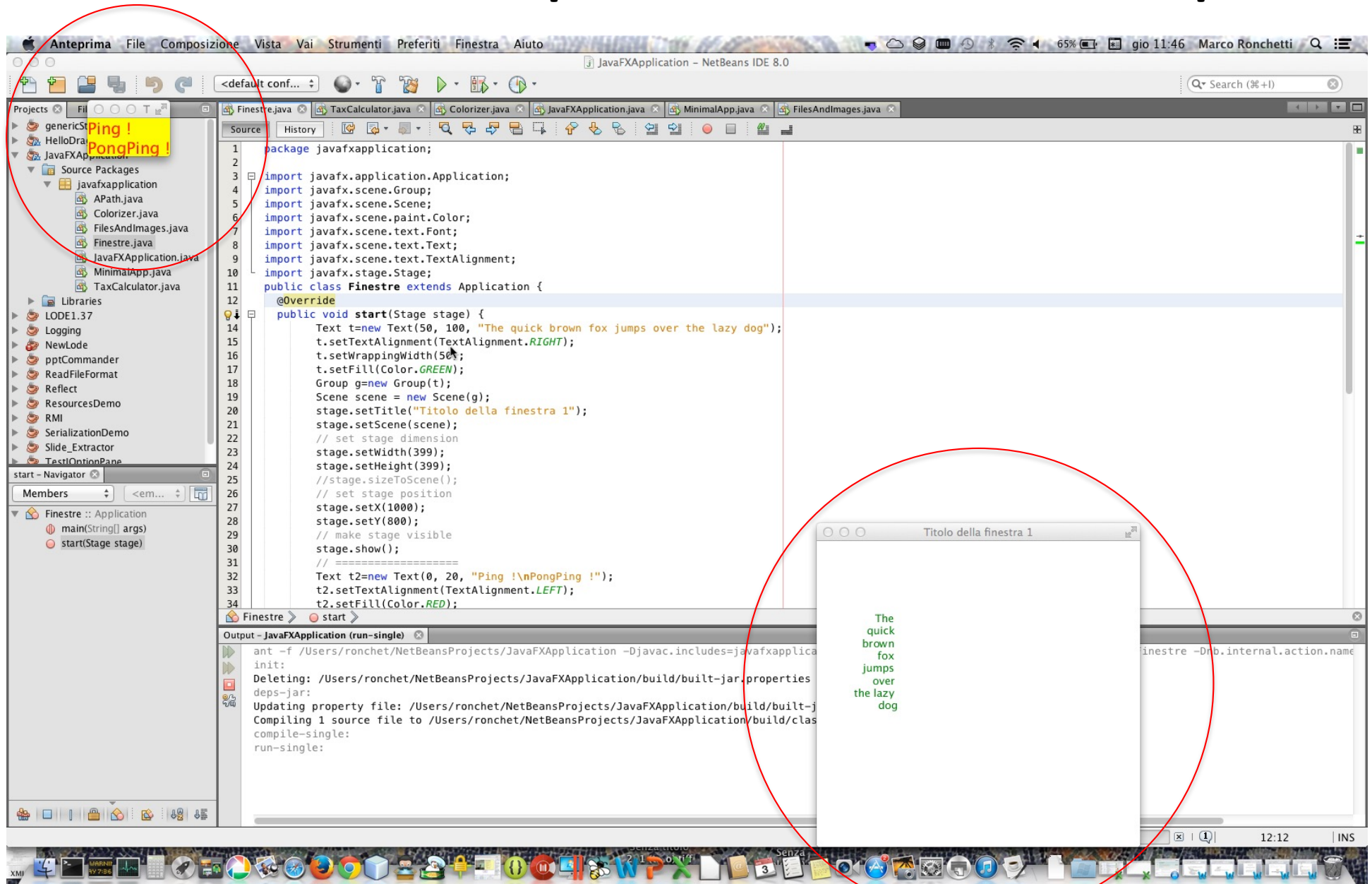
```
public class FilesAndImages extends Application {  
    public void start(Stage stage) {  
        FileChooser fileChooser = new FileChooser();  
        fileChooser.setTitle("Carica un'immagine");  
        fileChooser.getExtensionFilters().addAll(  
            new FileChooser.ExtensionFilter("JPG", "*.jpg"),  
            new FileChooser.ExtensionFilter("PNG", "*.png")  
        );  
        String url = System.getProperty("user.home");  
        File f=new File(url);  
        fileChooser.setInitialDirectory(f); // bugged on MacOSX  
        File file = fileChooser.showOpenDialog(stage);  
        if (file == null) {  
            System.out.println("No file chosen");  
            System.exit(1);  
        }  
    }  
}
```

Image & File

```
Image image = new Image("file://" +  
    file.getAbsolutePath(), 500, 500, true, true);  
ImageView iw = new ImageView(image);  
Group root = new Group(iw);  
Scene scene = new Scene(root, 500,500);  
stage.setTitle(file.getName());  
stage.setScene(scene);  
stage.sizeToScene();  
stage.show();  
}  
  
public static void main(String[] args) {  
    Application.launch(args);  
}  
}
```



Esempio: Finestre multiple



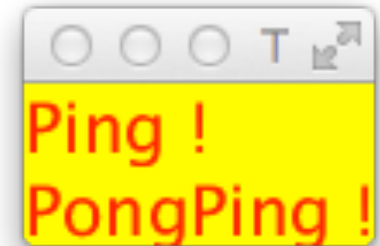
Finestre multiple: prima finestra

```
public class Finestre extends Application {  
    public void start(Stage stage) {  
        Text t=new Text(50, 100, "The quick brown fox jumps over  
            the lazy dog");  
        t.setTextAlignment(TextAlignment.RIGHT);  
        t.setWrappingWidth(50);  
        t.setFill(Paint.valueOf("GREEN"));  
        Group g = new Group(t);  
        Scene scene = new Scene(g);  
        stage.setTitle("Titolo  
            della finestra 1");  
        stage.setScene(scene);  
        // set stage dimension  
        stage.setWidth(399);  
        stage.setHeight(399);  
        // set stage position  
        stage.setX(400);  
        stage.setY(400);  
        // make stage visible  
        stage.show();  
    }  
}
```



Finestre multiple: seconda finestra

```
Text t2=new Text(0, 20, "Ping !\nPongPing !");
t2.setTextAlignment(TextAlignment.LEFT);
t2.setFill(Paint.valueOf("RED"));
t2.setFont(new Font(20));
Group g2 = new Group(t2);
Scene scene2 = new Scene(g2);
scene2.setFill(Paint.valueOf("YELLOW"));
Stage stage2 = new Stage();
stage2.setTitle("Titolo della finestra 2");
stage2.setScene(scene2);
stage2.sizeToScene();
stage2.setX(100);
stage2.setY(80);
stage2.show();
}
public static void main(String[] args) {
    launch(args);
}
}
```



Riassuntino...

Riassunto soluzione 1: casi semplicissimi.

A a=new A();
A ab=new B();
B b=new B();

```
class A {  
    public void g(A x) {System.out.println("called 0 on instance of A");}  
}  
class B extends A {  
    public void g(A x) {System.out.println("called 1 on instance of B");}  
    public void g(B x) {System.out.println("called 2 on instance of B");}  
}
```

Risoluzione firma banale,
Tipo statico e tipo dinamico coincidono, nessun problema!

a.g(a); => A.g(A) => A.g(A)
b.g(a); => B.g(A) => B.g(A)
b.g(b); => B.g(B) => B.g(B)

called 0 on instance of A
called 1 on instance of B
called 2 on instance of B

Riassunto soluzione 2: casi semplici.

A a=new A();
A ab=new B();
B b=new B();

```
class A {  
    public void g(A x) {System.out.println("called 0 on instance of A");}  
}  
class B extends A {  
    public void g(A x) {System.out.println("called 1 on instance of B");}  
    public void g(B x) {System.out.println("called 2 on instance of B");}  
}
```

Risoluzione firma banale,
Nell'argomento tipo statico e runtime non coincidono ma il fatto non è rilevante,
Tipo statico e tipo dinamico coincidono, nessun problema!

a.g(ab); => A.g(A) => A.g(A) called 0 on instance of A
b.g(ab); => B.g(A) => B.g(A) called 1 on instance of B

Riassunto soluzione 3: dynamic binding.

```
A a=new A();  
A ab=new B();  
B b=new B();
```

```
class A {  
    public void g(A x) {System.out.println("called 0 on instance of A");}  
}  
class B extends A {  
    public void g(A x) {System.out.println("called 1 on instance of B");}  
    public void g(B x) {System.out.println("called 2 on instance of B");}  
}
```

Risoluzione firma banale,
Tipo statico e tipo dinamico NON coincidono, dynamic binding!

```
ab.g(ab); => A.g(A) => B.g(A) called 1 on instance of B  
ab.g(b);  => A.g(A) => B.g(A) called 1 on instance of B
```

Riassunto soluzione 4: Liskov

```
A a=new A();  
A ab=new B();  
B b=new B();
```

```
class A {  
    public void g(A x) {System.out.println("called 0 on instance of A");}  
}  
class B extends A {  
    public void g(A x) {System.out.println("called 1 on instance of B");}  
    public void g(B x) {System.out.println("called 2 on instance of B");}  
}
```

Nella risoluzione statica non trovo il metodo cercato, e devo ricorrere a Liskov
Poi nel binding dinamico non ho complicazioni..

a.g(b); => A.g(B) => A.g(A) => **A.g(A)** called 0 on instance of A

Riassunto soluzione 5: Liskov + Dynamic Binding.

```
A a=new A();  
A ab=new B();  
B b=new B();
```

```
class A {  
    public void g(A x) {System.out.println("called 0 on instance of A");}  
}  
class B extends A {  
    public void g(A x) {System.out.println("called 1 on instance of B");}  
    public void g(B x) {System.out.println("called 2 on instance of B");}  
}
```

Nella risoluzione statica non trovo il metodo cercato, e devo ricorrere a Liskov
Poi nel binding dinamico devo adattare la soluzione trovata all'overriding.

ab.g(b); => A.g(B) => A.g(A) => **B.g(A)** called 1 on instance of B