

Tutorato Informatica III B

a. a. 2020-2021

Dott. Andrea Bombarda

Ing. Marco Radavelli

Prof.ssa P. Scandurra

- Software Distribution
- Event-based Java component model for Desktop UIs

Outline

- Software Distribution (Java):
 - Java Conventions
 - Java Deployment
 - Eclipse Tricks
 - Versioning
- Event-based Java component model for Desktop UI

Java Conventions

Convenzioni

La leggibilità del codice e il rispetto di alcuni standard nella sua stesura è di fondamentale importanza nella realizzazione di un buon prodotto software.

E' importante **rispettare le convenzioni** univocamente riconosciute in quanto facilita la comprensibilità, il riuso e la manutenzione del codice.

Elementi interessati dalle convenzioni

- Organizzazione dei File
- Package
- Blocchi
- Convenzioni sui nomi
- Commenti
- JAVA DOC

Organizzazione dei File

- Ogni file sorgente Java contiene una singola classe pubblica o un'interfaccia.
- Se ci sono classi private o interfacce associate, esse possono essere inserite nello stesso file dopo la classe pubblica principale che dà il nome al file.
- Tutti i file sorgenti Java devono avere la seguente struttura:
 - Commenti iniziali (autore, versione, data copyright,...)
 - Package e import
 - Dichiarazione della classe (o dell'interfaccia)

Package e Import

- La definizione del codice occupa la prima riga del codice:
package java.awt;
- Le righe successive prevedono l'import dei package necessari per la classe:
import java.awt.peer.CanvasPeer;
- Il primo campo del nome di un package univoco deve essere minuscolo e deve essere uno dei nomi dei domini di livello più alto:
edu, gov, mil, net, org
- Oppure le due lettere che descrivono una nazione come specificato nello standard ISO 3166, 1981 (it, uk, ch, fr, de,...)

Classi e Interfacce

1. Documentazione della classe o dell'interfaccia (`/** ... */`)
2. Dichiarazione della classe o dell'interfaccia
3. Commento generale all'implementazione, se necessario (`/* ... */`)
4. Class (static) variables: prima quelle pubbliche, poi quelle protette, poi quelle a livello package (senza modificatori) e per ultime le private
5. Instance variables: seguono lo stesso ordine delle class variables
6. Costruttori
7. Metodi: Dovrebbero essere raggruppati per funzionalità e non per scope o visibilità. L'obiettivo è quello di far comprendere il codice nel modo più facile e rapido

Dichiarazione delle variabili

- È consigliato dichiarare una variabile per linea, questo incoraggia i commenti.
- Inizializzare le variabili dove vengono dichiarate
- Posizionare le dichiarazioni all'inizio dei blocchi.

ESEMPIO:

SI:

```
for (int i=0; i<8; i++) {  
    double angolo = 2*Math.PI/8*i;  
    [...]  
}
```

EVITARE:

```
for (i=0;i<8;i++) {  
    [...] // some instructions  
    double angolo;  
    [...]  
    angolo=  
        2*Math.PI/8*i;  
    [...]  
}
```

Java Blocks

```
if (testScore >= 70) {  
    if (studentAge < 10) {  
        System.out.println("You did a great job");  
    } else {  
        System.out.println("You did pass"); //test score >= 70  
    } //and age >= 10  
} else { //test score < 70  
    System.out.println("You did not pass");  
}
```

Names

- I package hanno sempre il primo campo tutto minuscolo, i successivi campi sono a scelta.
- Classi e interfacce dovrebbero avere nomi che rappresentano sostantivi, in maiuscolo e possibilmente espressivi delle operazioni fornite. Ogni parola deve iniziare con la lettera maiuscola:

```
class Raster  
class VectorialImage
```

- I nomi dei metodi dovrebbero essere verbi con la prima lettera minuscola e le lettere interne maiuscole (secondo la cosiddetta “camel case notation”):

```
run();  
runFast();
```

Names

- Le variabili devono avere nomi significativi (eccetto per le temporanee: i, j, k, m, n...) e seguire la stessa convenzione dei metodi (prima lettera minuscola).

```
int size;  
myCalc myCalculator;
```

- Le variabili costanti devono essere espresse completamente in maiuscolo usando il carattere '_' come separatore:

```
static final int MIN_WIDTH = 4;  
static final int MAX_WIDTH = 999;
```

Commenti al codice

E' importantissimo che il codice sia leggibile anche a distanza di tempo e da parte di persone diverse dall'autore (per essere più facilmente comprensibile e modificabile).

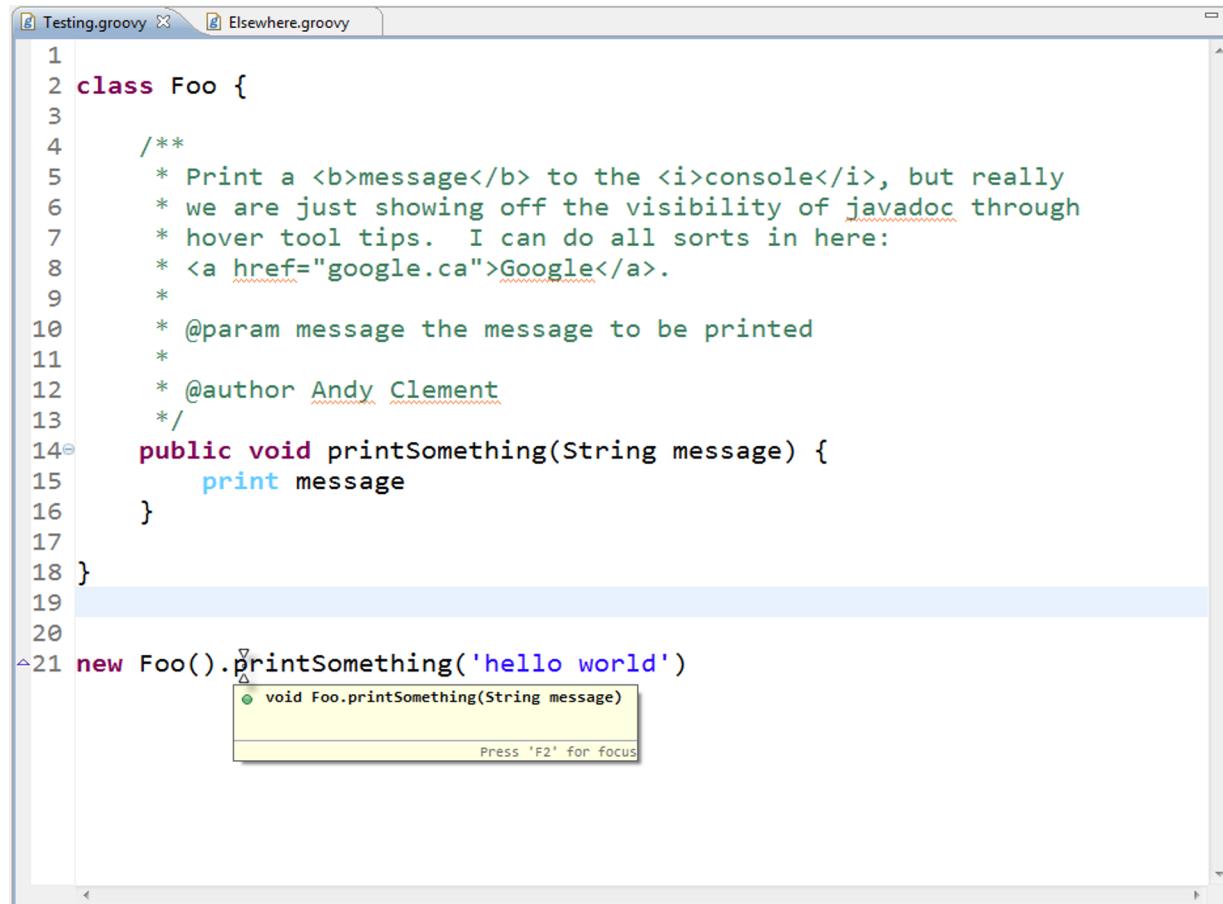
Per questo è necessario inserire nel codice commenti significativi, che spieghino le funzionalità dei vari metodi, le scelte fatte e quant'altro l'autore ritiene utile per la comprensione del programma.

```
/* Block comment */
import java.util.Date;
/**
 * Doc comment here for SomeClass
 * @version 1.0
 */
public class SomeClass { // some comment
    private String field = "Hello World";
    private double unusedField = 12345.67890;
    private UnknownType anotherString = "AnotherString";
    public SomeClass() {
        //TODO: something
        int localVar = "IntelliJ"; // Error, incompatible types
        System.out.println(anotherString + field + localVar);
        long time = Date.parse("1.2.3"); // Method is deprecated
    }
}
```

JAVAdoc

- *JavaDoc* nacque come strumento interno utilizzato dai ricercatori della Sun che stavano lavorando alla creazione del linguaggio Java e delle sue librerie
- La grande mole di sorgenti spinse alcuni membri del team a creare un programma per la **generazione automatica di documentazione HTML** (formato conosciuto, veloce da leggere, facilmente indicizzabile)
- Serviva un sistema automatico (per gestire la quantità di riferimenti incrociati che ci sono fra le classi e evitare gli errori di battitura).
- *JavaDoc* nacque quindi **per permettere ai programmatore di inserire dei frammenti HTML nei commenti** (ignorati quindi dal compilatore)

JAVAdoc



The screenshot shows a code editor window titled "Testing.groovy". The code defines a class "Foo" with a single method "printSomething". The JavaDoc comments for "printSomething" include a href tag pointing to "google.ca". A tooltip is displayed over the line "new Foo().printSomething('hello world')", showing the method signature "void Foo.printSomething(String message)". The tooltip also includes the instruction "Press 'F2' for focus".

```
1
2 class Foo {
3
4     /**
5      * Print a <b>message</b> to the <i>console</i>, but really
6      * we are just showing off the visibility of javadoc through
7      * hover tool tips. I can do all sorts in here:
8      * <a href="google.ca">Google</a>.
9      *
10     * @param message the message to be printed
11    *
12    * @author Andy Clement
13   */
14  public void printSomething(String message) {
15      print message
16  }
17
18 }
19
20
21 new Foo().printSomething('hello world')
```



A tooltip box is displayed, containing the following text:
void Foo.printSomething(String message)
Press 'F2' for focus

JAUTODOC

<http://jautodoc.sourceforge.net/index.html>

- JAutoDoc consente di:
 - Completare la documentazione esistente
 - Aggiungere altra documentazione senza modificare quella esistente
 - Sovrascrivere la documentazione esistente
- È possibile definire la visibilità dei campi da commentare, specificare filtri e template, inserire header nei file.

Per l'utilizzo di JAutoDoc

<http://jautodoc.sourceforge.net/index.html#usage>

- Preferenze documentazione:
 - Window -> Preferences -> java -> JAutoDoc
- Generazione documentazione:
 - Click destro su file o progetto -> JAutoDoc -> Add Javadoc
- Inserimento Header (vuoto o definito da un template specifico)
 - Click destro su file o progetto -> JAutoDoc -> Add header
- Generazione Javadoc:
 - Click destro su progetto -> export -> java -> javadoc
 - Può essere necessario specificare il path a javadoc.exe (distribuito con la JDK)
 - Osserva anche le altre opzioni (fogli di stile, modalità di esportazione, path, filtri...)

Java Deployment

JAR

Un file JAR (Java ARchive) è un file che contiene le [classi](#), le immagini, e tutti i vari file di una applicazione [Java](#) o di una [applet](#), raccolti in un unico file e generalmente compressi.

Utilizzando un «Java Development Kit» (JDK), si ha sempre generalmente inclusa una utility chiamata «Jar».

L'utility Jar permette la creazione e l'estrazione di file da un singolo file JAR. In una soluzione complessa, l'applicazione Java può essere realizzata anche da un insieme di file JAR. Un package Java [open source](#) può anche essere distribuito come file JAR.

Eclipse Jar Deployment

- Eclipse permette di generare un archivio jar eseguibile partendo da un progetto eseguibile (dotato di metodo main):
 - Click pulsante destro su progetto -> export -> java -> runnable jar file
 - Specificare il launch configuration da usare
 - Specificare il path di destinazione
 - Specificare come trattare le eventuali librerie richieste dall'applicazione
- Una volta generato è possibile eseguire il jar con il comando
`java -jar nome_del_jar.jar`

Eclipse Tricks

Eclipse Tricks: una breve lista

Comando	Effetto
CTRL+SHIFT+O	Sistema automaticamente gli import (N.B. in caso di omonimi in diversi package, chiede conferma)
Click destro -> Refactor -> Rename	Rinomina qualsiasi cosa evidenziata (classe, variabile, metodo..) aggiornando in automatico le referenze in tutto il progetto
CTRL+A seguito da CTRL+I	Indenta correttamente il codice selezionato (CTRL+A selezionato tutto il codice)
F3	Va a definizione (di qualsiasi metodo/variabile/classe... evidenziata dal cursore)
Click destro -> References -> Project	Mostra tutti i punti in cui quella variabile/classe/metodo è referenziato
Source -> Generate Constructor using Fields	Genera automaticamente il costruttore
Source -> Generate Getters and Setters...	Genera automaticamente i metodi GET e SET

Versioning

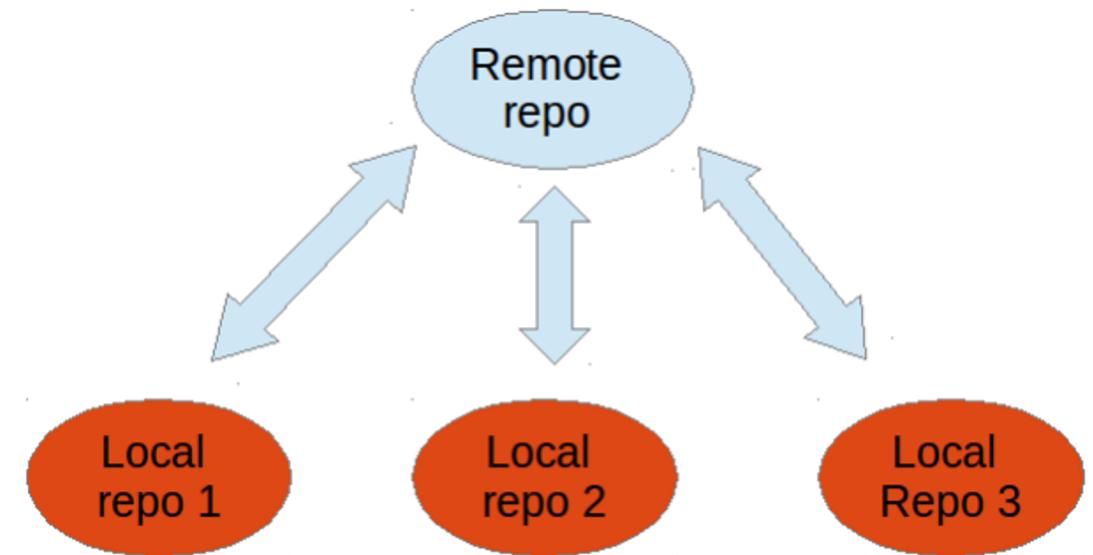
Controllo di Versione

- La gestione delle versioni consente di controllare e monitorare le modifiche fatte a file (anche, e soprattutto, file sorgenti). In particolare:
 - Quali modifiche sono state fatte? Quando? Da chi?
 - Tornare alla precedente versione
 - Quale codice era presente alla versione X.Y?
- I primi tool sono comparsi negli anni '70 e non sono più usati.
- In tempi più recenti sono stati sviluppati tool quali: RCV, CVS, Microsoft Source Safe, PVCS, Version Manager e altri
- Attualmente i tool più diffusi sono: **Subversion (svn)**, Mercurial e **Git**.

La logica alla base del Controllo di Versione

Il controllo di versione permette lo sviluppo collaborativo:

- Ogni sviluppatore lavora su una copia in locale del repository.
- Quando uno sviluppatore ha terminato le modifiche, sincronizza con il repository condiviso.
- Se l'ultima versione del repository locale prima delle modifiche è la stessa del repository remoto: viene fatto upload delle modifiche verso il repository remoto (Operazione di *commit*);
- Altrimenti (*se altri sviluppatori hanno sincronizzato prima di lui*), occorre fare un «update» per allineare il «Local repo» all'ultima versione «Remote repo» (risolvendo eventuali conflitti), e poi si può procedere all'upload (Commit).



SVN

- **Subversion** (noto anche come **svn**, che è il nome del suo *client* a riga di comando) è un sistema di controllo versione progettato da CollabNet Inc. con lo scopo di essere il naturale successore di CVS, oramai considerato superato.
- Useremo prima **svn** perché semplice, integrato in molti IDE e molto diffuso.
- Un'alternativa valida risulta essere **GIT** (anch'essa trattata di seguito).

Concetti fondamentali

- **Repository:** deposito gestito da un programma dove i file vengono memorizzati, indicizzati e gestiti. Solitamente raggiungibile via rete.
- **Local Copy:** copia locale dei file presenti sul repository.
- **Checkout:** ottieni una copia locale dei file dal repository.
- **Add:** aggiungi un file al version control (locale). Da questo momento in poi le modifiche al file saranno tracciate ed, eventualmente, sincronizzate con il repository remoto.

Concetti fondamentali

- **Commit:** invia al repository remoto le modifiche al version control locale.
- **Update:** aggiorna la copia locale con l'ultima versione dei file presenti sul repository
- **Tag/Branch:** etichetta una particolare release dell'insieme dei file in un certo momento

Tool SVN

- Tool grafici:
 - **Tortoise SVN** per Windows: aggiunge comandi al menu tasto destro di Explorer.
 - **SCPlugin per MacOS**: aggiunge comandi al menu tasto destro di Finder.
- Plugin Eclipse:
 - **Subversive SVN Team Provider**: include gli strumenti di gestione SVN in Eclipse e offre una speciale perspective per la gestione dei commit e per consultare lo storico modifiche.

Comandi SVN da console

Dato un repository on line:

1. Checkout

```
svn checkout [LOCATION] [PATH]
```

2. Modifica uno o più file

3. Aggiungi un nuovo file al controllo di versione

```
svn add [NOME DEL FILE]
```

In questo modo il file è aggiunto al controllo di versione ma non è ancora caricato sul repository remoto.

4. Esegui un commit:

```
svn commit -m «Messaggio di Commit»
```

In questo modo il file è caricato sul repository remoto.

5. Aggiorna eventuali modifiche fatte da altri utenti:

```
svn up
```

Memento

- Assicurarsi sempre di avere l'ultima versione
- Fare commit frequenti in modo da circoscrivere gli eventuali conflitti
- SVN ammette commenti vuoti ai commit ma documentare ciascun commit è fortemente consigliato (*anche per capire in che punto ripristinare una eventuale versione precedente!*)

Memento

- Non caricare versioni non stabili. Ad esempio, se gli unit test venivano eseguiti correttamente prima delle modifiche, dovranno farlo anche dopo.
- Carica solo i file necessari per la compilazione:
 - Sorgenti, header, file di configurazione, script.
- Non caricare, se possibile, librerie, file compilati, binari, .dll o .so, documentazione generata.
 - Non sono necessari, occupano spazio, provocano conflitti e non possono essere analizzati da SVN.

Conflitti e problemi

- SubVersion è in grado di accorgersi se le modifiche effettuate da diversi utenti si sovrappongono. In tal caso ci segnala il conflitto e ci chiede di risolverlo.
- Se il file è di tipo mergeable (quindi non un file binario) ci viene ancora più incontro, inserendo nel file stesso dei marcatori che evidenziano il conflitto, come nell'esempio qui riportato:

```
<<<<< .mine codice copia locale codice copia locale =====  
codice copia sul repository codice  
copia sul repository codice copia sul repository >>>>> .r2
```

La parte superiore mostra la versione locale del contenuto, mentre la parte inferiore mostra la versione del repository.

Conflitti e problemi

Vengono inoltre creati nella nostra copia di lavoro altri tre **file temporanei**:

- *nomefile.mine* – il nostro file locale, con le ultime modifiche che abbiamo cercato di caricare
- *nomefile.rBASE* – il file presente nella copia “nascosta”, quindi prima delle modifiche locali
- *nomefile.rHEAD* – l’ultimo file presente sul repository

BASE e HEAD vengono sostituiti dai corrispondenti numeri di versione.

Per risolvere il conflitto abbiamo quindi diverse strade:

- unire a mano le modifiche, sfruttando i marcatori inseriti da SubVersion
- sovrascrivere il file che crea conflitto con uno dei tre file temporanei sopra elencati
- usare il comando *svn revert* per annullare tutte le modifiche locali

Se scegliamo la via manuale, dovremo poi avvertire SVN dell’avvenuta risoluzione usando il

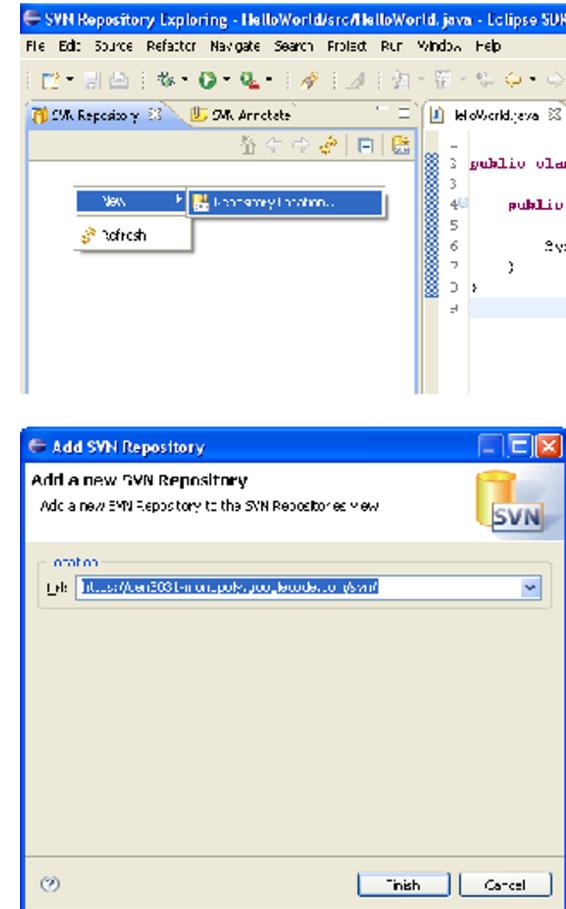
svn resolved nomefile

che ovviamente in TortoiseSVN diventa l’opzione **resolved** disponibile cliccando con il tasto destro sul file in questione.

Subversive: Eclipse integration

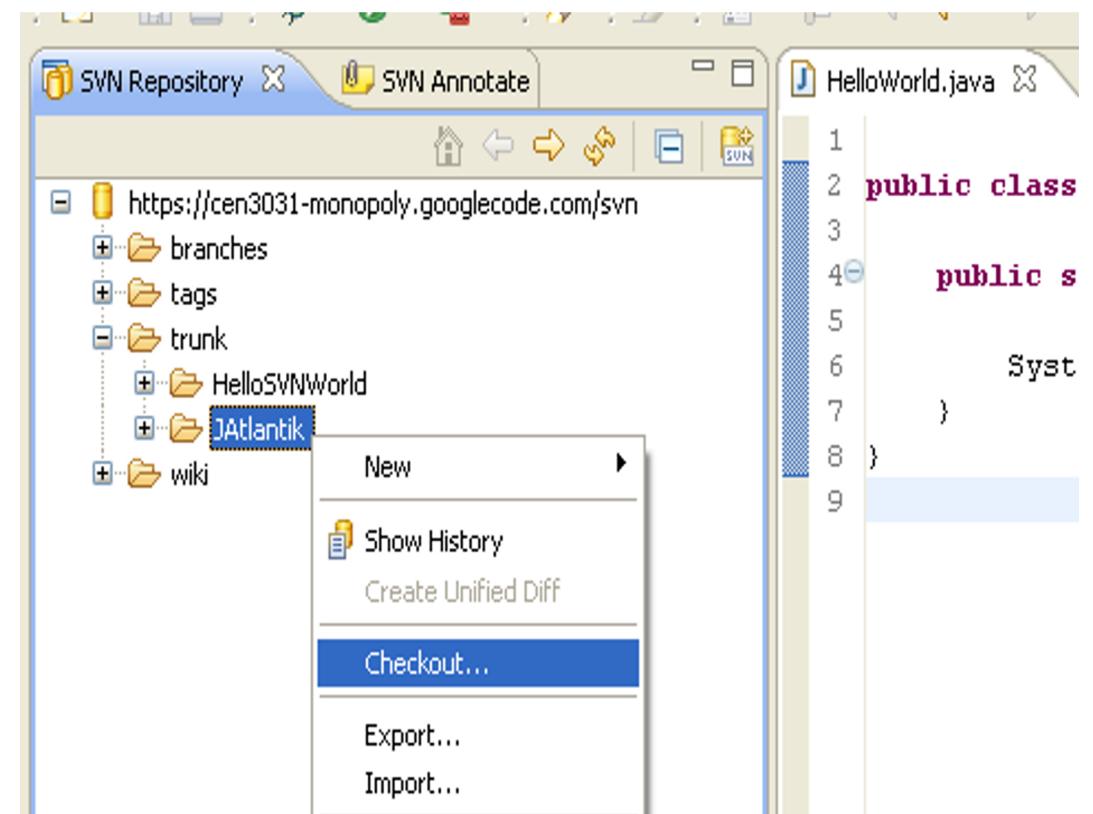
Il progetto è già sul server SVN

1. SVN perspective
2. Click tasto destro sulla finestra
3. Repository Location
4. Inserisci indirizzo ed eventualmente le credenziali d'accesso



Subversive: Eclipse integration

1. Scegli un progetto e fai checkout.
2. Ora il progetto è nel workspace locale.
3. È possibile modificare, eliminare, aggiungere file e fare commit, diff, update e risolvere conflitti dal menu «Team» raggiungibile con il tasto destro del mouse.



Comandi GIT da console



Dato un repository on line:

1. Clone (= Checkout di SVN): crea il repository locale

```
git clone [LOCATION]
```

2. Modifica uno o più file

3. Aggiungi un nuovo file al controllo di versione

```
git add [NOME DEL FILE]
```

In questo modo il file è aggiunto al controllo di versione ma non è ancora caricato sul repository remoto. Nel nome del file sono ammessi anche i caratteri jolly

4. Esegui un commit:

```
git commit -m «Messaggio di Commit»
```

In questo modo il file NON è ancora caricato sul repository remoto. Per effettuare il caricamento è necessario il comando

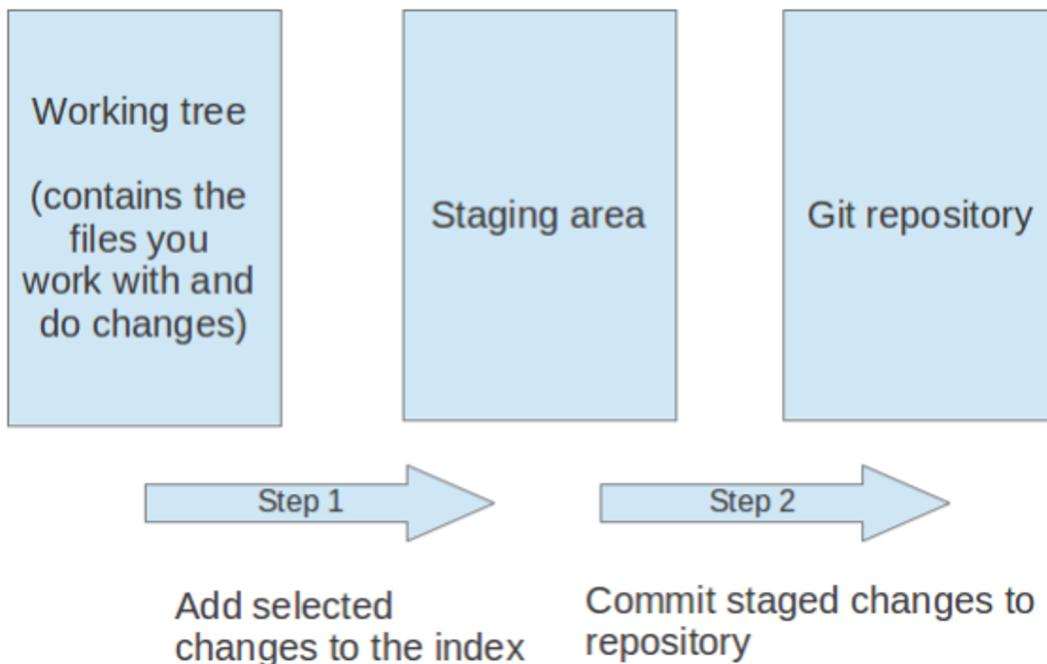
```
git push
```

5. Aggiorna eventuali modifiche fatte da altri utenti:

```
git pull
```

Il comando segnala anche in automatico eventuali conflitti

Comandi GIT da console



- Il comando **add** aggiunge i file alla “staging area”.
- Il comando **commit** allinea ciò che è nella staging area, nel repository (in locale).
- Il comando **push** fa l’*upload* del repository locale verso quello remoto, rendendolo disponibile anche agli altri sviluppatori (che posso “scaricarlo” con il comando **pull**).
- Prima di effettuare un “pull”, è possibile accantonare tutte le modifiche effettuate localmente (per esempio perché si è deciso che non sono più necessarie, o superate completamente da commit di altri sviluppatori), con il comando “**stash**”.

Git – prova pratica

- Esistono diversi servizi per l'hosting di repository Git disponibili in internet. Fra i più utilizzati vi sono **GitHub** e **Bitbucket** (per le nostre prove possiamo usare GitHub).
- In Eclipse esiste un client Git come plugin, chiamato Egit (*vedasi guida per l'installazione e l'uso*).

Esercizio per testare il corretto funzionamento di Git dopo averlo installato:

1. Registrarsi con l'email dell'università a <http://www.github.com> : dà diritto a creare repository privati.
2. Creare un nuovo repository.
3. Eseguire un push in quel repository, di un progetto di Eclipse, utilizzando il prompt dei comandi (*occorre posizionarsi nella root del progetto*).
4. Modificare un file, per esempio Main.java, ed eseguire i comandi:
 - `git add Main.java`
 - `git commit -m "mia modifica"`
 - `git pull`
 - Se non ci sono conflitti, proseguire con: `git push`

Git e JAutoDoc – prova pratica 2

Esercizio n°2 per testare il corretto funzionamento di Git dopo averlo installato:

1. Creiamo un progetto Java in Eclipse
2. Aggiungiamo una classe a piacere con almeno due campi ed un metodo (ad esempio due numeri ed un metodo che restituisce la loro somma)
3. Aggiungiamo i commenti in JavaDoc
4. Generiamo in automatico la documentazione JavaDoc tramite JAutoDoc
5. Facciamo il commit sul repository appena creato

Event-based component model for Desktop UI with Java

Swing API

- Swing è l'API (Application Programming Interface) Java che permette la creazione di interfacce visuali.
- Si basa sul modello MVC (Model/View/Controller)
- Caratteristiche principali:
 - Altamente personalizzabile ed estendibile.
 - Multi-piattaforma.
 - Bassi requisiti computazionali.

Alternative a Swing

Esistono diverse alternative a Swing. Le principali sono:

- **JFC/AWT (Abstract Window Toolkit)**
 - La accenneremo ma non la useremo
 - Implementazione «vecchia», risalente a Java 1.0
- **SWT (Standard Widget Toolkit)**
 - API grafica sviluppata dal team di Eclipse
 - Da molti ritenuta superiore (per funzionalità) a Swing.
 - Non è ancora uno standard vero e proprio
 - Non è implementato su tutte le piattaforme

Storia

- Nel 1996 Netscape ha introdotto una libreria GUI detta **IFC** (**Internet Foundation Class**)
 - Tale libreria «disegna» gli elementi di interfaccia dentro una finestra vuota.
 - Al SO è richiesta solo la costruzione delle finestre e la possibilità di disegnare su esse.

In questo modo i widget (elementi grafici) di IFC si presentano e comportano nello stesso modo su qualsiasi SO

- La libreria IFC è stata perfezionata dalla collaborazione con SUN ed in seguito ribattezzata **Swing**.

Swing fa parte di un framework più grande chiamato JFC (Java Foundation Class) che comprende anche

- Java2D
- API per il drag'n'drop
- API per l'accessibilità
- Molto altro ancora...

Swing non è un completo sostituto di AWT: alcune funzionalità delle GUI sono ancora gestite tramite AWT (eventi, layout, ecc...)

Gerarchia della libreria SWING

La libreria Swing è contenuta nel package javax.swing (javax è il prefisso di tutte le librerie che non sono il «core» di Java ma estensioni).

I nomi delle classi Swing iniziano tutti con una «j» minuscola: questo li distingue dalle controparti AWT. Per esempio:

SWING	AWT
jPanel	Panel
jTextArea	TextArea

Frame

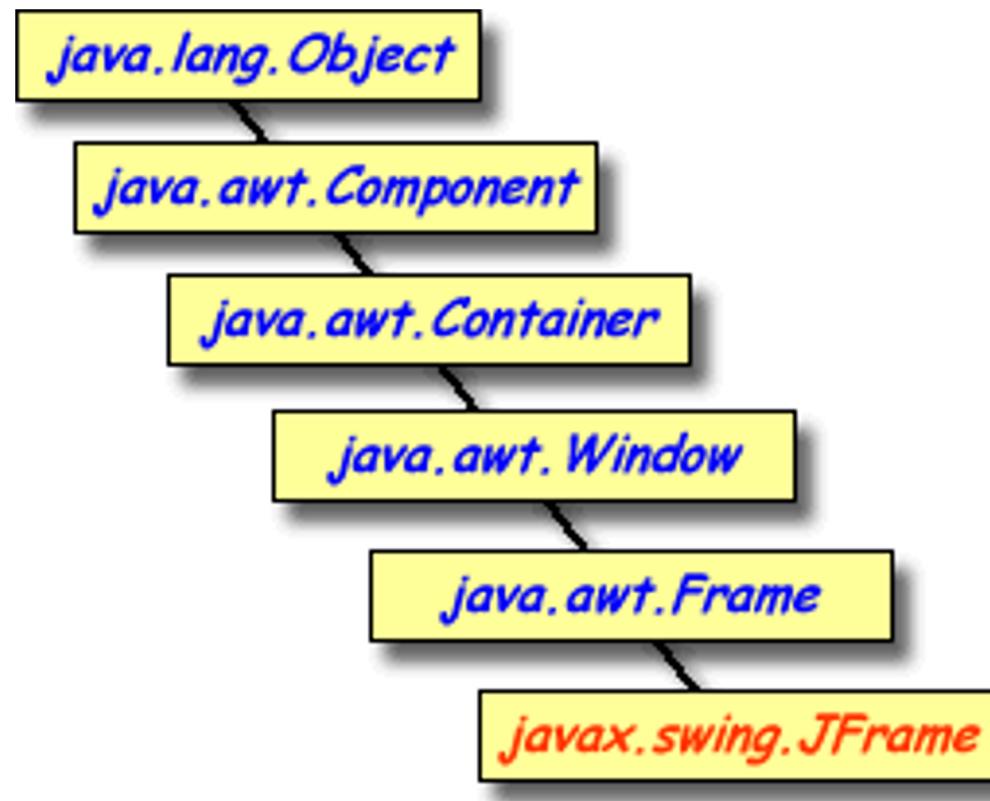
I **Frame** (cornice, intelaiatura, scheletro) sono la «struttura» che contiene un'applicazione grafica.

Un Frame fornisce un rettangolo «decorato» il cui compito consiste nel contenere tutto quello che l'applicazione deve mostrare all'utente.

È un oggetto molto complesso e ricco di opzioni, anche se fortunatamente la libreria Swing lo rende semplice da usare. I Frame offrono in automatico una serie di funzionalità base:

- Pulsanti di riduzione a icona, massimizzazione e chiusura
- Riposizionamento e ridimensionamento tramite mouse
- Barra del titolo
- Icona dell'applicazione

Conosciamo quali metodi eredita un frame....



awt.Window

- È un Container che può apparire sullo schermo come entità propria ma senza bordi, barre o controlli.
- Possiede metodi per gestirne la posizione e la visibilità:
 - void *setVisible(boolean b)*: mostra/nasconde la finestra (ereditato da *Component*).
 - void *setLocation(int x, int y)*: posiziona la finestra alle coordinate indicate.
 - void *setSize(int w, int h)*: dimensiona la finestra.
 - void *toFront()*, void *toBack()*: aggiusta l'ordine di comparsa relativamente ad altre finestre.

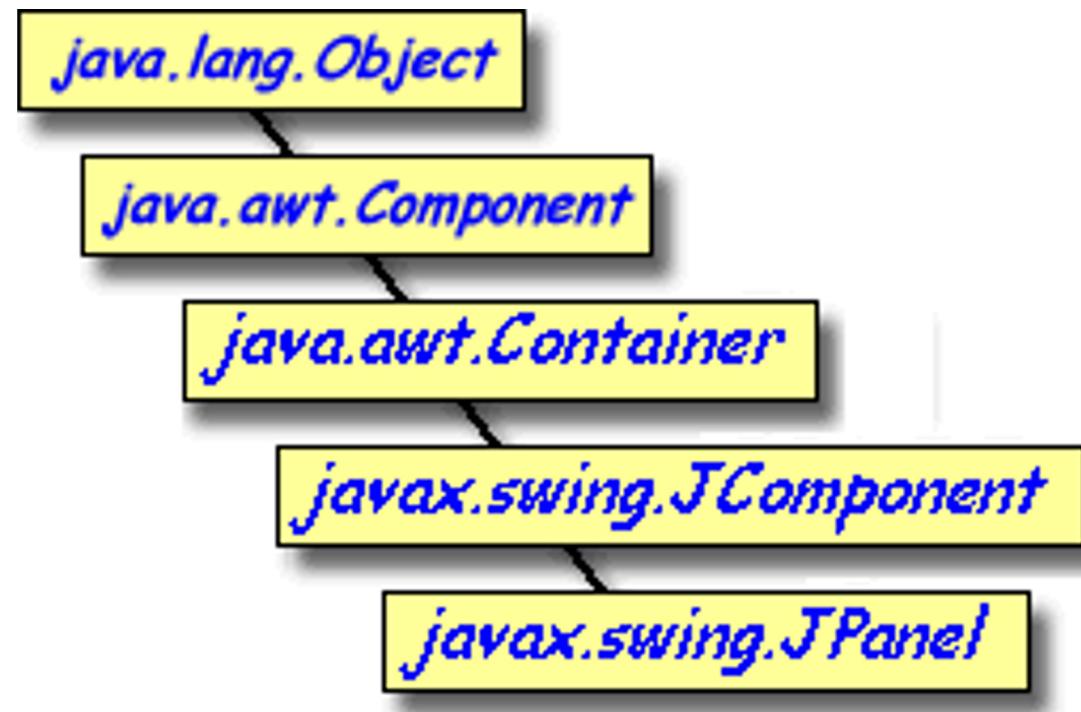
awt.Frame

- Un Frame è una Window a cui sono stati aggiunti bordi, titolo, pulsanti di controllo.
- Oltre ai metodi di window, un frame possiede metodi per gestire queste aggiunte:
 - void *setTitle*(String str): imposta un titolo
 - void *setResizable*(boolean b): imposta la “ridimensionabilità” di una finestra
 - void *setIconImage*(Icon icon): imposta l'icona della barra del titolo

javax.swing.JFrame

- Si tratta dell'effettivo Frame che verrà utilizzato nelle applicazioni grafiche che tratteremo.
- Esso è un Frame AWT a cui la libreria Swing aggiunge una serie di metodi importanti (anche per correggere malfunzionamenti di AWT).
- Un esempio di funzionalità aggiuntiva è quella dello **status**, ovvero la funzionalità che permette di ridurre ad icona o massimizzare l'applicazione:
 - **void *setExtendedState(int state)***: riduce a icona, massimizza e visualizza in modalità normale.
 - **void *setState(int state)***: analogo, ma non massimizza.

`javax.swing.JComponent` ← `swing.JPanel`



javax.swing.JComponent

- La classe JComponent introduce molte delle funzionalità legate a Swing.
- La più importante è la possibilità, per un elemento, di essere disegnato attraverso il metodo:
 - **void *paintComponent(Graphics g)***: disegna dentro un *JComponent*
- È la classe base per quasi tutti i componenti Swing.
- Per poter usare un componente che eredita da JComponent, occorre posizionare il componente in un container la cui radice sia uno Swing Container (JFrame, JDialog, JApplet).

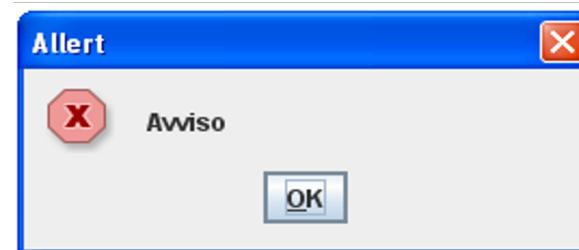
javax.swing.JPanel

- È una delle più semplici componenti Swing
- Serve come area rettangolare sulla quale:
 - Inserire altri pannelli Swing
 - Disegnare

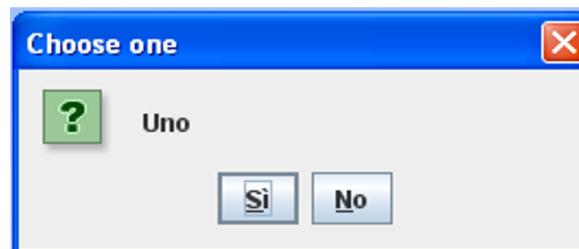
javax.swing.JOptionPane

- La classe JOptionPane è la classe usata per implementare finestre di dialog:

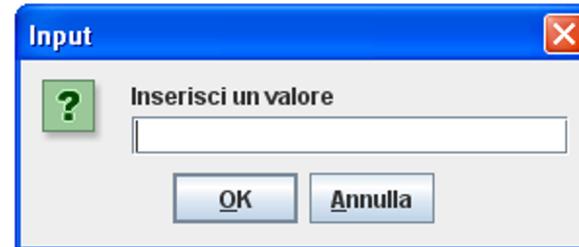
```
JOptionPane.showMessageDialog(null,  
    "Avviso", "Alert",  
    JOptionPane.ERROR_MESSAGE);
```



```
JOptionPane.showConfirmDialog(null,  
    "Uno", "Choose  
    one", JOptionPane.YES_NO_OPTION);
```



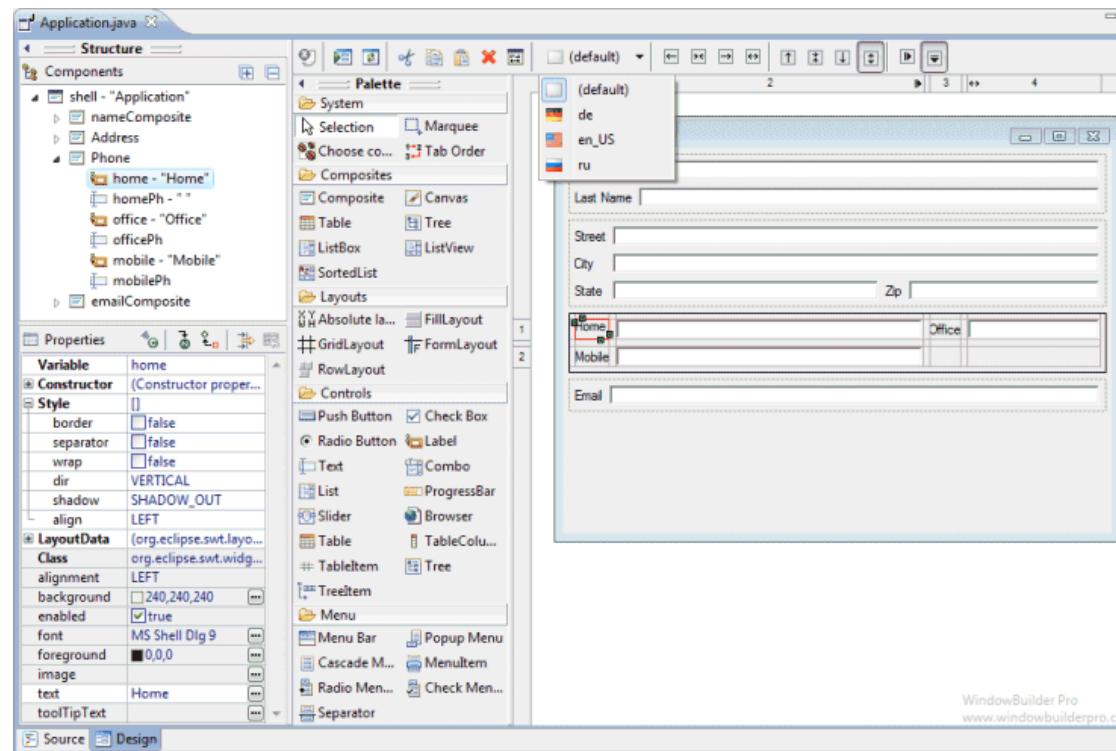
```
String inputValue =  
JOptionPane.showInputDialog ("Inserisci  
un valore");
```



WindowBuilder - Eclipse

<https://www.eclipse.org/windowbuilder/>

- **WindowBuilder** è un designer per SWT e Swing, integrato in Eclipse, che facilita la creazione di GUI Java, tramite l'utilizzo del drag-and-drop di componenti all'interno di un frame.



Java2D e Graphics

- La classe Graphics e il metodo paintComponent() consentono di disegnare e/o scrivere.
- Il metodo paintComponent() è ereditato da JComponent.
- Affinché ogni componente faccia ciò che si desidera, il metodo va ridefinito.

```
public void paintComponent(Graphics g)
{
    //Per ereditare le operazioni standard
    super.paintComponent(g);

    //Seguono istruzioni specifiche di disegno
    // per questo componente.
}
```

Graphics

- *Graphics* è una classe astratta
- Essa viene specializzata da:
 - i vari SO in funzione delle loro caratteristiche di disegno
 - le varie Component che hanno diritto ad un contesto grafico a seconda delle necessità e dei limiti di ciascuna componente
- Si possono così programmare operazioni grafiche su tutte le componenti

- Chi chiama *paintComponent*?
 - Viene chiamato automaticamente ogni qualvolta sia necessario. **Non deve essere chiamato manualmente!**
- Come forzare la chiamata del metodo *paintComponent*?
 - void *repaint()*: forza il ridisegno della finestra
- Quali azioni attivano il *paintComponent*?
 - Il metodo viene attivato da tutte le operazioni che provocano un ridisegno della finestra (ridimensionamento della finestra, riduzione ad icona, massimizzazione, sovrapposizione di finestre, ecc.

- Per lavorare il *paintComponent* ha bisogno di un oggetto *Graphics* (contesto grafico)
 - Esso è la sua “memoria” e il suo “libretto di istruzioni” per eseguire i comandi di disegno
- Un *Graphics* viene creato dal *JComponent* appena deve essere disegnato la prima volta
- *Graphics* mantiene le seguenti informazioni:
 - l'oggetto sul quale si disegna
 - il sistema di coordinate adottato per disegnare
 - il colore di foreground (colore con cui disegnare)
 - il font usato per le stringhe e le sue proprietà
 - il “clip” (ritaglio)
 - la modalità di disegno (Paint o XOR)
 - la modalità di trasparenza e di blending
 - Ecc..

Sistema di riferimento per gli oggetti Graphics



Per gli oggetti racchiusi in un rettangolo l'origine è sempre il vertice in alto a sinistra del rettangolo.

Le “misure” sono sempre in pixel.

Classi per oggetti Graphics

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

- Alcune classi importanti quando si lavora con oggetti Graphics sono:
 - Color (*definizione con RGB, costanti della classe*): può essere utilizzata per impostare il colore di sfondo, il colore in primo piano o un colore generale.
 - Font (*definizione della dimensione del carattere, font-family, stile del testo*): può essere utilizzata per definire il font del contesto grafico.
 - Disegni vari (*forme elementari, linee, spezzate, poligoni*).
 - Toolkit e Image: per la gestione di immagini.