2018
Programmazione a oggetti
parte 3

Giorgio Bruno

Dip. Automatica e Informatica
Politecnico di Torino
email: giorgio.bruno@polito.it

Argomenti

Metodi clone e getClass della classe Object

Classe Random

Formattazione

Classe Scanner

File, package java.io

Nuove classi e interfacce per l'accesso al file system, package java.nio.file

Espressioni regolari

Date

Thread

Pattern

Interfacce grafiche: Swing, JavaFX

Metodi clone e getClass della classe Object

Il metodo clone fornisce la copia di un oggetto in maniera efficiente. La classe i cui oggetti possono essere clonati deve implementare l'interfaccia **Cloneable** (che è vuota) e ridefinire *clone* come *pubblico* (da protected). La clonazione può essere superficiale o profonda.

Il metodo getClass fornisce un oggetto di tipo Class mediante il quale si possono ottenere informazioni sulle proprietà (attributi, metodi e costruttori) della classe con il supporto del package java.lang.**reflect**.

Esempi di clonazione

```
public class Point implements Cloneable {
   private int x,y;
   public int getX() {return x;}
   public int getY() {return y;}
   public Point(int x, int y) {this.x = x; this.y = y;}
   public void move(int x, int y) {this.x = x; this.y = y;}
   public String toString(){return "x = " + x + " y = " + y;}

    public Object clone(){
          try {return super.clone();}
          catch (CloneNotSupportedException e) {return null;}
    }
}
```

clonazione superficiale; solo valori

L'interfaccia **Cloneable** (in java.lang) è vuota. Per convenzione le classi che la implementano devono ridefinire clone (che è protected) con un metodo pubblico.

Se manca la clausola implements **Cloneable** l'istruzione return super.clone() lancia l'eccezione CloneNotSupportedException.

Clonazione di Rectangle

```
public class Rectangle implements Cloneable{
   private int width = 20; private int height = 10;
   private Point origin = null;
   public Rectangle() {origin = new Point(0, 0);}
   public Rectangle(Point p, int w, int h) {
                        origin = p; width = w; height = h;}
   public String toString(){return origin.toString()+ " w = "
                                                  + width + " h = " + height;}
   public void move(int x, int y) {origin.move(x,y);}
   public int area() {return width * height;}

   public Object clone(){
         try {Rectangle r = (Rectangle) super.clone();
         r.origin = (Point)origin.clone();return r;}
         catch (CloneNotSupportedException e) {return null;}
   }
}
```

clonazione profonda: si clona anche il punto origin.

Programma di test

```
public static void main(String[] args) {
   Point p1 = new Point(23, 94);
   Point p2 = (Point) p1.clone();    p2.move(10,20);
   System.out.println(p1); // x = 23 y = 94
   System.out.println(p2); // x = 10 y = 20
   Rectangle r1 = new Rectangle(p1, 100, 200);
   Rectangle r2 = (Rectangle) r1.clone();    r2.move(10,20);
   System.out.println(r1); // x = 23 y = 94 w = 100 h = 200
   System.out.println(r2); // x = 10 y = 20 w = 100 h = 200
   // i rettangoli hanno origini separate
}
```

Esempi di uso di Class

```
Point p1 = new Point(10,20);
Rectangle r1 = new Rectangle(p1,100,200);

Class<? extends Rectangle> c1 = r1.getClass();
// r1 potrebbe puntare ad un oggetto di una classe derivata da Rectangle

System.out.println(c1.getName());
        // nome del package.Rectangle
```

Nota: il metodo **getSimpleName**() dà soltanto il nome della classe.

Metodo newInstance

Class<T> offre il metodo

public T **newInstance**() throws InstantiationException,
IllegalAccessException

mediante il quale si può istanziare un oggetto con il costruttore privo di parametri.

```
Rectangle r2 = c1.newInstance();
System.out.println(r2);
```

```
x = 0 y = 0 w = 20 h = 10
```

Metodo forName

Il metodo statico **forName**

static Class<?> forName(String className)

dà l'oggetto Class in base al nome della classe preceduto dal nome del package.

Se Rectangle si trova nel package p

Nota: si può generare un oggetto dato il nome della classe!

```
String className = "p.Rectangle";
@SuppressWarnings("unchecked")
Class<Rectangle> c2 =
        (Class<Rectangle>) Class.forName(className);
// il cast serve perché non si sa a priori di quale classe si tratti.
Rectangle r3 = c2.newInstance();
System.out.println(r3);
```

x = 0 y = 0 w = 20 h = 10

Classe Random

Si trovano nel package java.util.

Class Random

Le istanze di Random producono sequenze di numeri pseudocasuali.

2 costruttori: Random() e Random(long seed)

Alcuni metodi

IntStream **ints**(long n, int l1, int l2) fornisce uno stream di n interi compresi tra i due limiti (l1 incluso, l2 escluso)

Metodi analoghi per LongStream e DoubleStream.

int **nextInt**(), long nextLong(), double nextDouble

danno il numero successivo

int nextInt(int l)

dà l'intero successivo compreso tra 0 (incluso) e l (escluso).

Esempi

```
Random r = new Random();
IntStream stream = r.ints(10, 100, 200);
int[] v = stream.toArray();
System.out.println (Arrays.toString(v));

for(int i = 0; i < 3; i++) {
System.out.println(r.nextInt());
}
for(int i = 0; i < 3; i++) {
System.out.println(r.nextInt(10));
}
```

```
[107, 132, 106, 114, 145, 127,
175, 182, 137, 191]

1228546375
256012486
613500919

6
7
4
```

Formattazione

Il metodo **format** di String (e PrintWriter) permette di separare la parte fissa di una stringa dalla parte variabile.

Esempio

```
String autore = "falco"; String titolo = "rosso";
String editore = "deltaplano";  int pagine = 100; double prezzo = 15.5;
String formato = String.format("%s, %s, %s, %d, %.2f%n",
    autore,titolo,editore,pagine,prezzo);
System.out.print(formato); // falco, rosso, deltaplano, 100, 15,50
```

Definizione di format

```
static String     format(String format, Object... args)
static String     format(Locale l, String format, Object... args)
```

Nell'esempio la stringa format è: "%s, %s, %s, %d, %.2f%n"

Simboli per indicare gli elementi

```
String formato = String.format("%s, %s, %s, %d, %.2f%n",
        autore,titolo,editore,pagine,prezzo);
```

%s sta per una stringa

%d sta per un numero intero

%.2f sta per un numero decimale con 2 cifre dopo la virgola (o il punto)

%n significa a capo

Il risultato **falco, rosso, deltaplano, 100, 15,50** è formato dalla concatenazione di

una stringa ", "

una stringa ", "

una stringa ", "

un numero intero ", "

un numero decimale con 2 cifre dopo la virgola o il punto (dipende dal Locale)

a capo

Formattazione con Locale

La separazione tra la parte intera e quella decimale è data dalla virgola in italiano e dal punto in inglese.

Le varie convenzioni si possono indicare con un oggetto di classe Locale nel modo seguente

```
String formato = String.format(Locale.UK,"%s, %s, %s, %d, %.2f%n",
   autore,titolo,editore,pagine,prezzo);
System.out.print(formato); // falco, rosso, deltaplano, 100, 15.50
formato = String.format(Locale.ITALY,"%s, %s, %s, %d, %.2f%n",
   autore,titolo,editore,pagine,prezzo);
System.out.print(formato); // falco, rosso, deltaplano, 100, 15,50
```

Se si omette il parametro Locale è seguita la convenzione di default.

La classe Locale si trova in java.util.

Classe Scanner

Uno scanner spezza una stringa in tanti token che si possono poi leggere con i metodi **next** (con parametro di vari tipi) e **hasNext**.

I token sono sottostringhe separate da delimitatori che si possono definire, mediante **useDelimiter**, con espressioni regolari come ",\\s*" (virgola seguita da spazi). Il delimitatore di default è lo spazio.

Esempio

```
String linea = "falco, rosso, deltaplano, 100, 15.5";
Scanner s = new Scanner(linea);
s.useDelimiter(",\\s*");
while (s.hasNext()) System.out.println(s.next());
s.close();
```

```
falco
rosso
deltaplano
100
15.5
```

Note: l'ultimo elemento è il prezzo del libro.

Tutti i token sono letti come stringhe.

Class Scanner

Metodi usati

Scanner(String source)
void close() // **chiudere dopo l'uso**
boolean hasNext()
String next() // può lanciare l'eccezione NoSuchElementException
double nextDouble()
int nextInt()
+ next per gli altri tipi numerici
Scanner useDelimiter(String pattern)
Scanner useLocale(Locale locale)

Lettura con conversione automatica

```
String linea = "falco, rosso, deltaplano, 100, 15";
Scanner s = new Scanner(linea);
s.useDelimiter(",\\s*");
String autore = null; String titolo = null; String editore = null;
int pagine = 0; float prezzo = 0;
        autore = s.next(); titolo = s.next();
        editore = s.next(); pagine = s.nextInt();
        prezzo = s.nextFloat();
s.close();
String formato = String.format("%s, %s, %s, %d, %.2f%n",
                    autore,titolo,editore,pagine,prezzo);
System.out.print(formato); // falco, rosso, deltaplano, 100, 15,00
```

Controllo errori

```
String linea = "falco, rosso, deltaplano, 100, 15";
//manca deltaplano,
Scanner s = new Scanner(linea);
s.useDelimiter(",\\s*");
String autore = null; String titolo = null; String editore = null;
int pagine = 0; float prezzo = 0;
        autore = s.next(); titolo = s.next();
        editore = s.next(); pagine = s.nextInt();
        prezzo = s.nextFloat();
s.close();
```

Exception in thread "main" java.util.NoSuchElementException

Controllo errori con try/catch

Si scriva il metodo

Libro genLibro(String linea) //factory method

il cui compito è di leggere la stringa e dare come risultato un nuovo libro oppure null se la stringa è errata.

Es.

String linea = "falco, rosso, 100, 15";

System.out.println(genLibro(linea));

Risultato

linea errata: falco, rosso, 100, 15 // stampa effettuata da genLibro

null

Controllo errori con try/catch

```
public static Libro genLibro(String linea) {
Scanner s = new Scanner(linea);
Libro libro = null;
String autore = null; String titolo = null; String editore = null;
int pagine = 0; float prezzo = 0;
try {
        s.useDelimiter(",\\s*");
        autore = s.next(); titolo = s.next();
        editore = s.next(); pagine = s.nextInt();
        prezzo = s.nextFloat();
        libro = new Libro(autore, titolo, editore, pagine, prezzo);
} catch(Exception ex) {System.out.println("linea errata: " + linea);}
finally {s.close();}
return libro;
}
```

Lettura di numeri decimali

Se la convenzione corrente è quella italiana e la stringa contiene invece un numero che ha il punto come separatore tra la parte intera e quella decimale, occorre indicare allo scanner di usare il locale appropriato, Locale.UK.

Infatti il metodo genLibro dà un errore se la linea è
"falco, rosso, deltaplano, 100, **15.50**";

Per impostare il locale si usa il metodo useLocale
quindi: **s.useLocale(Locale.UK);**

```
public static Libro genLibro(String linea) {
Scanner s = new Scanner(linea); …
try {
        s.useDelimiter(",\\s*"); s.useLocale(Locale.UK);
        …
} catch(Exception ex) {System.out.println("linea errata: " + linea);}
finally {s.close();}
return libro;
}
```

File

Classi principali

File testuali	FileReader, BufferedReader FileWriter, PrintWriter
File binari	FileInputStream, DataInputStream FileOutputStream, DataOutputStream
Per System.in	InputStreamReader, BufferedReader
Eccezioni	IOException, EOFException, FileNotFoundException

Try con risorse

File

I file sono visti come stream (sequenze) di caratteri o di dati in formato binario.

Tipi di stream:

byte stream: letti e scritti a byte

character stream: letti e scritti a caratteri

testuali (line-oriented): letti e scritti a linee; sono usati buffer per migliorare l'efficienza

data stream: contengono valori e stringhe in formato **binario**; la struttura deve essere nota a priori.

Le operazioni principali sono: **apertura** di file in lettura oppure in scrittura, **chiusura** di file, **lettura** di elemento, **scrittura** di elemento.

Le operazioni possono sollevare eccezioni di tipo **IOException** (sottoclasse di Exception); IOException è checked.

Le classi si trovano nel package java.io.

Byte stream

lettura e scrittura di un byte alla volta

```
//copia di file: da libri.txt a libriOut.txt

public static void byteStreams() { //try with resources
        try (FileInputStream in = new FileInputStream("libri.txt");
             FileOutputStream out = new FileOutputStream("libriOut.txt");)
             {int c; while ((c = in.read()) != -1) out.write(c); // -1 end of file
        } catch(IOException e) {System.out.println(e.getMessage());}
}
```

apertura lettura/scrittura: **FileInputStream / FileOutputStream**

chiusura: *close*; automatica grazie all'uso di try with resources.

Try con risorse

```
try (  // risorse che saranno chiuse automaticamente all'uscita dalla try
)
{
} catch …
```

Senza risorse, nell'esempio precedente serve una clausola finally

```
finally {
        if (in != null) in.close();
        if (out != null) out.close();
}
```

Character stream

lettura e scrittura di un carattere alla volta

```
public static void characterStreams() { //try with resources
        try (FileReader in = new FileReader("libri.txt");
              FileWriter out = new FileWriter("libriOut.txt");)
        {int c; while ((c = in.read()) != -1) out.write(c);
        }catch(IOException e) {System.out.println(e.getMessage());}
}
```

codifica dei caratteri: Unicode a 16 bit

apertura lettura/scrittura: **FileReader / FileWriter**

File testuali

```
public static void fileTestuali() {
try (BufferedReader in = new BufferedReader(new FileReader("libri.txt"));
      PrintWriter out = new PrintWriter (new FileWriter("libriOut.txt"));)
      {String line;
           while ((line = in.readLine()) != null) {
            out.println(line);
           }
      } catch(IOException e) {System.out.println(e.getMessage());}
…
```

lettura e scrittura di una linea alla volta

apertura e lettura:	FileReader e BufferedReader
apertura e scrittura:	FileWriter e PrintWriter

Bufferizzazione: **BufferedReader**; si può usare anche BufferedWriter.
Operazioni: *readLine*, print, println.
La lettura è completa quando *readLine dà null*.

File testuali

```
try(BufferedReader in = new BufferedReader(new FileReader("libriOut.txt")))
{List<String> lista = in.lines().collect(toList());
            System.out.println(lista);
 }
   catch(IOException e) {System.out.println(e.getMessage());}
}
```

Un'unica operazione per leggere tutte le linee

Il metodo **lines**() di BufferedReader fornisce uno stream di stringhe.
L'esempio raccoglie le linee in una lista di stringhe.

Dato il file di input:
falco, rosso, deltaplano, 100, 15.5
rondine, blu, caravella, 200, 20

La stampa produce:
[falco, rosso, deltaplano, 100, 15.5, rondine, blu, caravella, 200, 20]

File binari (scrittura)

```
public static void dataStreams() {//legge un file testuale e scrive un file binario
try(BufferedReader in = new BufferedReader(new FileReader("libri.txt"));
DataOutputStream out = new DataOutputStream (new FileOutputStream("libriOut.bin"));)
{       String line;
        while ((line = in.readLine()) != null) {
                Scanner s = new Scanner(line); s.useDelimiter(",\\s*"); s.useLocale(Locale.US);
                String autore = s.next(); String titolo = s.next(); String editore = s.next();
                int pagine = s.nextInt(); float prezzo = s.nextFloat(); s.close();

                out.writeUTF(autore); out.writeUTF(titolo); out.writeUTF(editore);
                out.writeInt(pagine); out.writeFloat(prezzo);
        }} catch(IOException e) {System.out.println(e.getMessage());}
```

Apertura in scrittura: **FileOutputStream e DataOutputStream**

Copia da file testuale a file binario: da libri.txt a libriOut.bin

Scrittura: **writeUTF per stringhe, writeInt, writeFloat**

UTF = Unicode Transformation Format

File binari (lettura)

```
// legge il file binario e lo stampa
try(DataInputStream in = new DataInputStream (new
        FileInputStream("libriOut.bin"));)
   {while (true) {
        String autore = in.readUTF(); String titolo = in.readUTF();
        String editore = in.readUTF(); int pagine = in.readInt();
        float prezzo = in.readFloat();
        System.out.format("%s, %s, %s, %d, %.2f%n",
                autore,titolo,editore,pagine,prezzo);
        // oppure
        System.out.format(Locale.US,
                "%s, %s, %s, %d, %.2f%n",autore,titolo,editore,pagine,prezzo);
    }
 } catch(EOFException e) {}
   catch(IOException e) {System.out.println(e.getMessage());}
```

Lettura: readUTF, readInt, readFloat.

Apertura in lettura: **FileInputStream e DataInputStream**

Data stream (lettura di file binario)

```
try (DataInputStream in =
        new DataInputStream (new FileInputStream("libriOut.bin"));)
    {while (true) {
        …
    }
 } catch(EOFException e) {}
   catch(IOException e) {System.out.println(e.getMessage());}
```

La lettura è completa quando è sollevata l'eccezione **EOFException**; in questo caso l'eccezione non indica un problema. EOFException è una sottoclasse di IOException.

Command line

L'esempio legge una stringa e la converte in maiuscolo e così via finché non legge end.

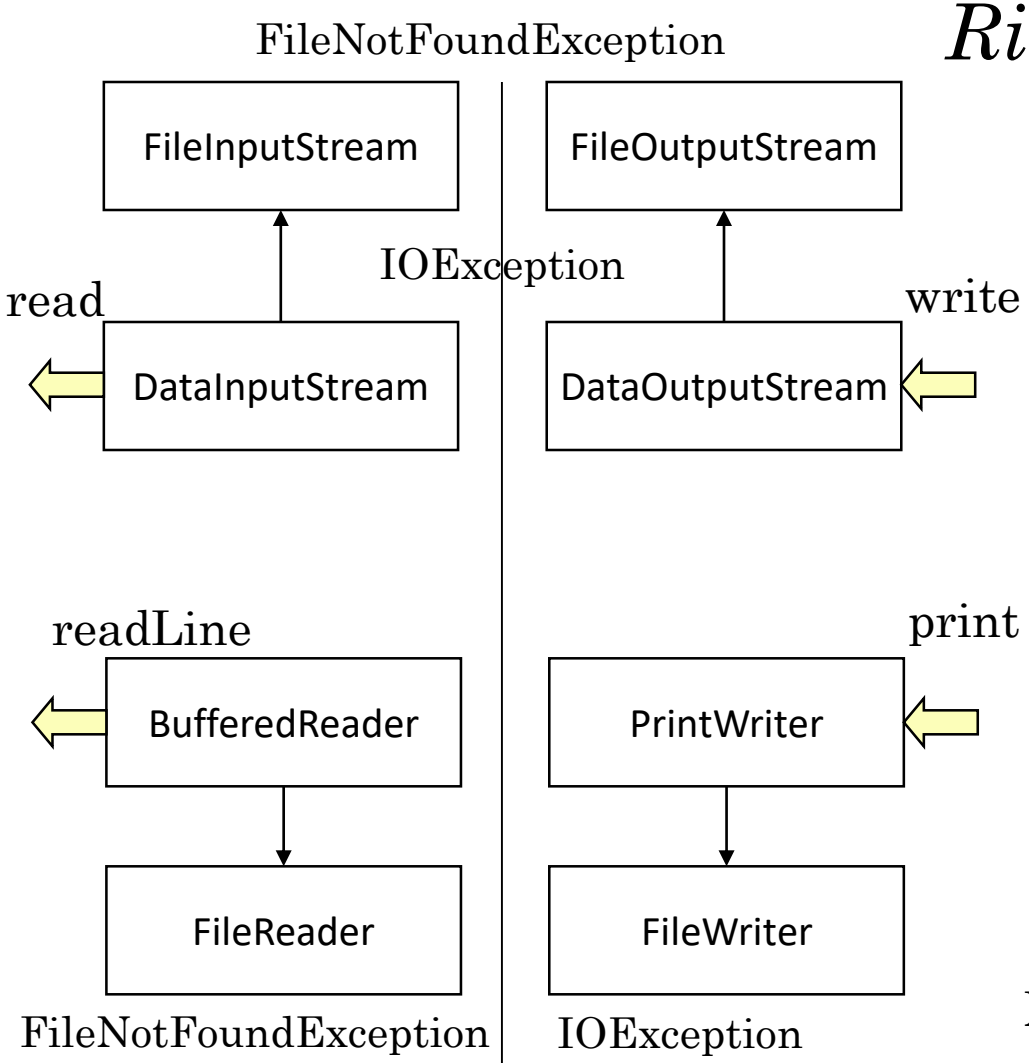
```
public static void commandLine() {
try (BufferedReader keyboard = new BufferedReader(
            new InputStreamReader(System.in));)
   {while (true) {
            System.out.print("input text: ");
            String l = keyboard.readLine();
            System.out.println(l.toUpperCase());
            if (l.equalsIgnoreCase("end")) break;
    }
}catch(IOException e) {System.out.println(e.getMessage());}
}
```

```
input text: hello
HELLO
input text: end
END
```

System.in è un byte stream che per le sue caratteristiche va letto mediante un **InputStreamReader** (ponte da byte stream e character stream).

System.out è un **PrintStream** (converte caratteri in bytes).

Riepilogo classi di IO



file binari

I/O di tipi primitivi

readInt, readDouble, readUTF

writeInt, writeDouble, writeUTF

file di caratteri

Note:

FileNotFoundException extends IOException

Esercizio

Leggere un file testuale saltando le linee errate.
Esempio

corretto
falco, rosso, deltaplano, 100, 15.5
rondine, blu, caravella, 200, 20

errato
file `libriErrati.txt`
falco, rosso, 100, 15.5 // manca deltaplano
rondine, blu, caravella, 200, 20

Soluzione

```
try (BufferedReader in =
        new BufferedReader(new FileReader("libriErrati.txt")))
  {String line; Scanner s = null;
  while ((line = in.readLine()) != null) {
        try {
              s = new Scanner(line);
              s.useDelimiter(",\\s*"); s.useLocale(Locale.US);
              String autore = s.next(); String titolo = s.next();
              String editore = s.next(); int pagine = s.nextInt();
              float prezzo = s.nextFloat();
              System.out.format("%s, %s, %s, %d, %.2f%n",
                    autore,titolo,editore,pagine,prezzo);
        } catch (Exception e) {System.out.println("linea errata: " +
                                        line);
        } finally {s.close();}
  }
} catch(IOException e) {System.out.println(e.getMessage());}
```

linea errata: falco, rosso, 100, 15.5
rondine, blu, caravella, 200, 20,00

Pattern

```
try(BufferedReader in =
        new BufferedReader(new FileReader("libriErrati.txt")))
  {String line; Scanner s = null;
  while ((line = in.readLine()) != null) {
        try {
              // analisi linea
        } catch (Exception e) {System.out.println("linea errata: " + line);
        } finally {s.close();}
  }
} catch(IOException e) {System.out.println(e.getMessage());}
```

Note:

blocchi try annidati

Nuove classi e interfacce per l'accesso al file system

Si trovano nel package **java.nio.file**.

Interfaccia Path

Classe Paths

Classe Files

Path e Paths

L'interfaccia **Path** rappresenta il percorso di un file o cartella.

La classe **Paths** offre il metodo statico **get** che fornisce il path data una stringa contenente il percorso del file o cartella.

Esempio

```
Path fileP =
Paths.get("G:\\workspaceLuna\\esempiTeoria\\pubblicazioni.txt");
```

Il metodo **getFileName** di Path dà come path il nome del file senza il percorso, ad es. pubblicazioni.txt.

Classe Files

Questa classe offre metodi statici per operazioni su file e cartelle.

Per leggere tutto un file testuale si possono usare i 2 metodi seguenti:

List<String> linee = Files.**readAllLines**(fileP); // fileP è un path

Stream<String> lines = Files.**lines**(fileP);

Per scrivere tutto un file testuale si può usare il metodo write:

Files.**write**(fileP1, linee); //fileP1 è un path.

Per ottenere lo stream dei path dei file di una cartella si usa il metodo **list**:

Stream<Path> cartelleS = Files.**list**(cartellaP); // cartellaP è un path

I metodi precedenti possono lanciare IOException.

Classe Files

Gli esempi seguenti mostrano come

1. convertire in maiuscolo le linee di un file
2. stampare i nomi dei file testuali (.txt) presenti in una cartella.

Esempio 1

G:\\esempio.txt

alfa
gamma
omega

G:\\esempioUC.txt

```
String path1 = "G:\\esempio.txt";
String path2 = "G:\\esempioUC.txt"; //UC = upper case
```

Esempio 1

```
import java.io.*; import java.nio.file.*;
import java.util.stream.*; import java.util.*;
…
public static void main(String[] args) {
// path dei file di input e output
String path1 = "G:\\esempio.txt";
String path2 = "G:\\esempioUC.txt"; //UC = upper case
```

G:\\esempio.txt

alfa
gamma
omega

G:\\esempioUC.txt

ALFA
GAMMA
OMEGA

Si vedranno due soluzioni

Esempio 1a

```
Path fileP1 = Paths.get(path1); Path fileP2 = Paths.get(path2);
List<String> linee = null;

//SOLUZIONE CON Files.readAllLines
try {
        linee = Files.readAllLines (fileP1);
        for (ListIterator<String> iter = linee.listIterator(); iter.hasNext();) {
                String s = iter.next().toUpperCase(); iter.set(s);}
        Files.write(fileP2, linee);
    } catch (IOException ex) {ex.printStackTrace();}
```

Si può usare un for compatto?

Note: **readAllLines** chiude il file.

Per sostituire la stringa corrente si genera quella nuova e si chiama il metodo set dell'iteratore.

Esempio 1b

```
Path fileP1 = Paths.get(path1); Path fileP2 = Paths.get(path2);
List<String> linee = null;

//SOLUZIONE CON Files.lines

try (Stream<String> lines = Files.lines(fileP1)) {
        linee = lines.map(String::toUpperCase).collect(toList());
        Files.write(fileP2, linee);
} catch(IOException e) {System.out.println(e.getMessage());}
```

Note

Si usa try with resources try (Stream<String> lines = Files.**lines**(fileP1)) per chiudere lo stream alla fine.

Esempio 2

```
Path cartellaP = Paths.get("G:\\workspaceLuna\\esempiTeoria");
try (Stream<Path> cartelleS = Files.list(cartellaP)) {
        List<String> nomi = cartelleS
        .map(Path::getFileName)
        // si ottiene come path il file name (senza percorso)
        .map(Path::toString) // si converte in stringa
        .filter(s -> s.endsWith(".txt"))
        // si tengono le stringhe che finiscono in .txt
        .collect(toList());
        System.out.println(nomi);
} catch(IOException e) {System.out.println(e.getMessage());}
```

Es. di risultato: [libriOut.txt, libriOut1.txt, libri.txt, pubblicazioni.txt, …]

Espressioni regolari

Struttura di un'espressione regolare

Metodi di String

Classi Pattern e Matcher

Espressioni regolari

Un'espressione regolare definisce un **pattern di caratteri** che può essere usato in vari modi, ad es. per suddividere una stringa in sottostringhe, per sostituire certe sottostringhe in una stringa, per validare una stringa.

Esempi

```
String separatore = "\\s*,\\s*"; //*spazi virgola *spazi
String stringa = "alfa, beta ,    gamma ,";
String[] sottostringhe = stringa.split(separatore);
System.out.println(Arrays.toString(sottostringhe)); // [alfa, beta, gamma]
String nuovaStringa = stringa.replaceAll(separatore, " ");
System.out.println("(" + nuovaStringa + ")"); // (alfa beta gamma )
String test = ",  ";
System.out.println(test.matches(separatore));  // true
```

Metodi di String: split, replaceAll, matches.

Forma di un'espressione regolare

Un'espressione regolare contiene caratteri e metacaratteri.

Esempi:

ab: il pattern costituito da 2 caratteri, a e b.

[ab]: 1 carattere, a o b.

[^ab]: 1 carattere diverso da a e b.

[a-g]: 1 carattere compreso tra a e g (inclusi).

\\d: una cifra.

\\D: un carattere diverso da una cifra.

\\s: uno spazio (oppure tab, carriage return).

\\S: un carattere diverso da uno spazio (oppure tab, carriage return).

\\w: una lettera o una cifra o "_".

\\W: un carattere diverso dai precedenti.

Metacaratteri (separati da virgole): [,], ^, -, \\.

Forma di un'espressione regolare

Quantificatori:

*	zero o più
+	1 o più
?	zero o uno
{n}	n volte,
{m, n}	da m a n volte.

Metacaratteri (separati da virgole): *, + ?, {, }.

Una parola corrisponde al pattern \\w+.

```
String test = "un'espressione regolare, oppure reg-ex";
String sep = "\\W+";
    \\W+ = sequenza di caratteri diversi da lettera, cifra o _
String [] sottostringhe = test.split(sep);
//[un, espressione, regolare, oppure, reg, ex]
```

Gruppi

Un gruppo di caratteri si indica tra **parentesi ()**.

Consideriamo un codice iban: 27 caratteri in 6 gruppi

es. IT 02 L 12345 01132 000056789012

```
String ibanRE = "([A-Z]{2})(\\d{2})([A-Z])(\\d{5})(\\d{5})(\\d{12})";
String iban = "IT02L123450113200056789012";
System.out.println(iban.matches(ibanRE)); //true
```

Note: ibanRE significa iban regular expression

Metodo di String: public boolean **matches**(String regex)

Gruppi

Formattazione dell'iban con 1 spazio tra gruppi adiacenti.

String format = "$1 $2 $3 $4 $5 $6";

I gruppi sono numerati da 1 e sono preceduti da **$**

String ibanFormattato = iban.replaceAll(ibanRE, format);

System.out.println(ibanFormattato);

//IT 02 L 12345 01132 000056789012

Note:

Nell'esempio, String replaceAll (String regex, String replacement) sostituisce con il format dato (replacement) tutte le sottostringhe di iban che corrispondono all'espressione regolare ibanRE (regex).

Match più efficienti con Pattern e Matcher

Le classi **Pattern** e **Matcher** si trovano nel package java.util.regex.

Un pattern contiene la forma compilata di un'espressione regolare.
Esempio
Pattern ibanP = Pattern.compile(ibanRE);
Si usa il metodo statico compile.

Matcher

Un matcher esegue operazioni su una sequenza di caratteri interpretando un pattern.

Con il metodo matcher di Pattern si genera un matcher per una data sequenza di caratteri, ad es. Matcher m = ibanP.matcher(iban);

Il metodo boolean **matches**() di Matcher
verifica la corrispondenza tra l'intera sequenza e il pattern.

Ad esempio
Pattern ibanP = Pattern.compile(ibanRE);
Matcher m = ibanP.matcher(iban);
boolean risultato = m.matches();

Esempio

Controllo sintattico del file libriDaValidare.

Esempio di contenuto del file:

falco, rosso, deltaplano, 100, 15.5

rondine, blu, caravella, 200, 20.0

aquila, nero, monociclo, , 12.50

Ogni linea ha 3 parole, un n. intero e un n. decimale; il separatore è una virgola seguita da spazi. Ogni linea inizia con una parola.

Si controlli il file e si stampino le linee accettate.

ESPRESSIONE REGOLARE

String sep = ",\\s*"; String parola = "\\w+";

String nIntero = "[0-9]+"; String nDecimale = "[0-9]+.[0-9]+";

String eRLibro = parola + sep + parola + sep + parola + sep + nIntero + sep + nDecimale;

Esempio

```
Pattern p = Pattern.compile(eRLibro);
Path fileP = Paths.get("libriDaValidare.txt");
List<String> linee = null;
try(Stream<String> lines = Files.lines(fileP)) {
        linee = lines
        .filter(l -> {return p.matcher(l).matches();})
        .collect(toList());
} catch(Exception e) {System.out.println(e.getMessage());}

System.out.println(linee);
// [falco, rosso, deltaplano, 100, 15.5, rondine, blu, caravella, 200, 20.0]
```

La linea errata è saltata.

Date

Il package java.time offre nuove classi per il trattamento delle date; in particolare:

LocalDate
LocalDateTime
ZonedDateTime
Period
Year

enum Month
enum DayOfWeek

Esempio

```
LocalDate oggi = LocalDate.now();
        System.out.println(oggi); // anno-mese-giorno  4 cifre, 2 cifre, 2 cifre
String oggiS = oggi.toString();
LocalDate oggi1 = LocalDate.parse(oggiS);
        System.out.println(oggi1);
        System.out.println(oggi1.getMonth());
        System.out.println(oggi.getDayOfWeek());
LocalDate natale = LocalDate.of(Year.now().getValue(), Month.DECEMBER, 25);
        System.out.println(natale);
System.out.println(natale.isAfter(oggi));
```

Confronto tra date con isAfter e isBefore.

```
2017-10-12
2017-10-12
OCTOBER
THURSDAY
2017-12-25
true
```

Esempio

```
LocalDateTime adesso = LocalDateTime.now();
        System.out.println(adesso);
LocalDate d1 = oggi.plusDays(20);
        System.out.println(d1);
Period p1 = Period.between(oggi, d1);
        System.out.println(p1);
        System.out.format("il periodo comprende %d giorni%n", p1.getDays());
        System.out.println(Arrays.toString(Month.values()));
        System.out.println(Arrays.toString(DayOfWeek.values()));
ZonedDateTime zdt = ZonedDateTime.now();
        System.out.println(zdt);
```

```
2017-10-12T19:16:22.710
2017-11-01
P20D
il periodo comprende 20 giorni
```

```
[JANUARY, FEBRUARY, MARCH, APRIL, MAY, JUNE, JULY, AUGUST, SEPTEMBER, OCTOBER, NOVEMBER, DECEMBER]
[MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY]
2017-10-12T19:16:22.726+02:00[Europe/Berlin]
```

Threads

Implementano elaborazioni concorrenti (almeno logicamente).
Occorre gestire la mutua esclusione e le sincronizzazioni.

Quando si pone in esecuzione un programma, si attiva un thread di default per il main.

Scrittura dei Threads

2 possibilità:

1. Si definisce una sottoclasse di Thread e si scrive la logica nel metodo **run** (che è chiamato allo **start** del thread).

2. Se una classe implementa l'interfaccia **Runnable** (ossia ha un metodo run) si può associare ad un thread un suo oggetto.

Class X implements Runnable { … }

Runnable r = new X(…);

Thread t = new Thread(r);

t.**start**();

```
public interface Runnable {

void run(); }
```

Classe Thread (java.lang)

Alcuni metodi:

public Thread(String name)
public Thread(Runnable target, String name)
public Thread(Runnable target) // nome generato automaticamente
public String getName()
public void interrupt()
public static void sleep(long millis) throws InterruptedException
public void start()

L'eccezione InterruptedException è lanciata quando il thread è interrotto mentre è in attesa o è sleeping.

Esempi

1. Due thread stampano 10 stringhe, ciascuno ad intervalli casuali <= 1 secondo.
2. Un thread continua a stampare stringhe e un altro (il main) lo interrompe dopo 10 sec.
3. Un produttore e due consumatori interagiscono tramite una coda di stringhe.
4. Uso di ArrayBlockingQueue; il package java.util.concurrent offre classi utili nella programmazione concorrente.
5. Esempio dei 5 filosofi.

Esempio 1

```
public class SimpleThread extends Thread {
    public SimpleThread(String name) {
        super(name); }
    public void run () {
        for (int i = 0; i < 10; i++) {
            System.out.println(i + " " + getName());
            try {sleep((long)(Math.random() * 1000));
            } catch (InterruptedException e) {}
        }
        System.out.println("DONE! " + getName());
    }
}
Programma di test
  new SimpleThread("Torino").start();
  new SimpleThread("Milano").start();
```

Due thread stampano 10 stringhe, ciascuno ad intervalli casuali <= 1 secondo.

```
0 Torino
0 Milano
1 Milano
1 Torino
2 Milano
2 Torino
3 Torino
4 Torino
3 Milano
5 Torino
4 Milano
5 Milano
6 Milano
6 Torino
7 Torino
8 Torino
9 Torino
7 Milano
8 Milano
DONE! Torino
9 Milano
DONE! Milano
```

Esempio 2

Un thread continua a stampare stringhe e un altro lo interrompe dopo 10 sec.

```
public class SimpleThread3 implements Runnable {
String name = "SimpleThread3";
String getName() {return name;}
    public void run () {
            int i = 0;
            try {while (true) {
                    System.out.println(i++ + " " + getName());
                    Thread.sleep((long)(Math.random() * 1000));
             }} catch (InterruptedException e) {}
            System.out.println("DONE! " + getName());}

public static void main(String[] args) throws InterruptedException{
            Runnable r = new SimpleThread3();
            Thread t = new Thread(r); t.start();
            Thread.sleep(10 * 1000); t.interrupt();
}}
```

```
0 SimpleThread3
1 SimpleThread3
2 SimpleThread3
3 SimpleThread3
4 SimpleThread3
5 SimpleThread3
6 SimpleThread3
7 SimpleThread3
8 SimpleThread3
9 SimpleThread3
10 SimpleThread3
11 SimpleThread3
12 SimpleThread3
13 SimpleThread3
14 SimpleThread3
15 SimpleThread3
16 SimpleThread3
17 SimpleThread3
18 SimpleThread3
19 SimpleThread3
20 SimpleThread3
21 SimpleThread3
22 SimpleThread3
DONE! SimpleThread3
```

Produttori e consumatori: requisiti

Esempio di produttori e consumatori che interagiscono tramite una coda di stringhe.

I produttori aggiungono stringhe lette da un array ricevuto come parametro; i consumatori tolgono stringhe. Produttori e consumatori hanno un nome e un intervallo massimo di azione; l'intervallo effettivo è un valore casuale tra 0 e l'intervallo massimo.

La coda ha una capacità: se è piena, il produttore aspetta; se è vuota, il consumatore aspetta. La coda registra il numero di attese dei produttori e quello dei consumatori.

I thread scrivono in un log ciò che hanno letto (scritto) dalla (nella) coda.

Produttori e consumatori

Problemi:

- accesso mutuamente esclusivo ad un metodo; metodo **synchronized**
- sospensione se una condizione risulta falsa; **wait**
- risveglio dei thread sospesi; **notify** o **notifyAll**

Il metodo wait può lanciare un'eccezione di tipo InterruptedException.

I metodi wait, notify e notifyAll sono definiti nella classe Object e sono final.

Metodi sincronizzati e guarded blocks

Mentre un thread esegue un metodo **synchronized** di un oggetto, gli altri thread che chiamano un metodo synchronized dello stesso oggetto sono bloccati.

Ogni oggetto ha un lock, chiamato *monitor*. Quando un thread chiama un metodo synchronized di un oggetto il cui lock è libero, il thread acquisisce il lock (che diventa impegnato) ed esegue il metodo. Se invece il lock è impegnato si accoda.

Quando un metodo synchronized termina, il lock diventa libero e può essere acquisito da un eventuale thread in coda.

Guarded blocks

Se occorre che una certa condizione sia vera e il thread la trova falsa, allora può eseguire una **wait**: come effetto il thread rilascia il lock e si sospende. Verrà poi risvegliato da una **notify** o **notifyAll** eseguita da un altro thread. Il thread risvegliato deve accertarsi che la condizione di prosecuzione sia verificata: in caso contrario chiamerà ancora la wait (di solito si usa un loop).

Coda

```
import java.util.*;
public class Coda {
        private int capacità; private int np, nc;
        LinkedList<String> coda = new LinkedList<>(); // per la remove
public Coda(int c) {capacità = c;}
public synchronized void aggiungi (String s) throws InterruptedException {
        while (coda.size() == capacità) {np++; wait();}
        coda.add(s); notifyAll();
}
public synchronized String togli () throws InterruptedException {
        while (coda.size() == 0) {nc++; wait();}
        String s = coda.remove(); notifyAll(); return s;
}
public synchronized String toString() {
        return np + " " + nc;
}
}
```

np (nc) = numero di volte in cui un produttore (consumatore) effettua una wait.

Consumatore

```
public class Consumatore implements Runnable {
        String nome; int intervallo; Coda q;
        StringBuilder log = new StringBuilder();
public Consumatore(String nome, int intervallo, Coda q) {
        this.nome = nome; this.intervallo = intervallo; this.q = q;}
public void run() {
        try {
        while (true){
                String s = q.togli(); log.append(" " + s);
                long p = (long)(Math.random() * 1000 * intervallo);
                Thread.sleep(p);
        }
        } catch (InterruptedException e) {}
        System.out.println(nome + " " + log);}
}
```

Produttore

```
public class Produttore implements Runnable {
        String nome; int intervallo; Coda q;
        StringBuilder log = new StringBuilder();
        String[] text;
public Produttore(String nome, int intervallo, Coda q, String[] text;) {
        this.nome = nome; this.intervallo = intervallo; this.q = q;
        this.text = text;}
public void run() {
        int i = 0;
        try {
        while (true){
                q.aggiungi(text[i]); log.append(" " + text[i]);
                i = (i+1) % text.length;
                long p = (long)(Math.random() * 1000 * intervallo);
                Thread.sleep(p);
        }
} catch (InterruptedException e) {}
System.out.println(nome + " " + log);}
```

Main

```
public static void main(String[] args) throws InterruptedException {
Coda q = new Coda(5);
String[] text =
"Nel mezzo del cammin di nostra vita mi ritrovai per una selva oscura".split(" ");
Thread p1 = new Thread(new Produttore("p1", 1, q, text));
Thread c1 = new Thread(new Consumatore("c1", 2, q));
Thread c2 = new Thread(new Consumatore("c2", 2, q));
p1.start(); c1.start(); c2.start();
Thread.sleep(20 * 1000);
p1.interrupt(); c1.interrupt(); c2.interrupt();
System.out.println(q);
}
```

c2 del di nostra vita ritrovai per oscura Nel del di vita per oscura del di vita
7 17
p1 Nel mezzo del cammin di nostra vita mi ritrovai per una selva oscura Nel mezzo del cammin di nostra vita mi ritrovai per una selva oscura Nel mezzo del cammin di nostra vita mi ritrovai per una selva
c1 Nel mezzo cammin mi una selva mezzo cammin nostra mi ritrovai una selva Nel mezzo cammin nostra

Uso di ArrayBlockingQueue

ArrayBlockingQueue implementa BlockingQueue.
Si trovano in *java.util.concurrent*.

Interface BlockingQueue<E>
void **put**(E e); inserisce l'elemento e aspetta se la coda è piena.
E **take**(); rimuove il primo elemento e aspetta se la coda è vuota.

Costruttore di ArrayBlockingQueue
public ArrayBlockingQueue(int capacity)

Produttore1

```
package esempiThreads;
import java.util.concurrent.*;
public class Produttore1 implements Runnable {
        String nome; int intervallo; BlockingQueue<String> q;
        StringBuilder log = new StringBuilder(); String[] text;
public Produttore1(String nome, int intervallo, BlockingQueue<String> q,
        String[] text) {this.nome = nome; this.intervallo = intervallo; this.q = q;
        this.text = text;}
public void run() {int i = 0;
try {
   while (true){
        q.put(text[i]); log.append(" " + text[i]); i = (i+1) % text.length;
        long p = (long)(Math.random() * 1000 * intervallo);
        Thread.sleep(p);
   }
} catch (InterruptedException e) {}
System.out.println(nome + " " + log);}
```

Consumatore1

```
package esempiThreads;
import java.util.concurrent.*;
public class Consumatore1 implements Runnable {
        String nome; int intervallo; BlockingQueue<String> q;
        StringBuilder log = new StringBuilder();
public Consumatore1(String nome, int intervallo, BlockingQueue<String> q) {
        this.nome = nome; this.intervallo = intervallo; this.q = q;}
public void run() {
try {
    while (true){
        String s = q.take(); log.append(" " + s);
        long p = (long)(Math.random() * 1000 * intervallo);
        Thread.sleep(p);
    }
} catch (InterruptedException e) {}
System.out.println(nome + " " + log);}
}
```

Main

```
public static void main(String[] args) throws InterruptedException {
BlockingQueue<String> q = new ArrayBlockingQueue<String>(5);
String[] text = "Nel mezzo del cammin di nostra vita mi ritrovai per una selva oscura".split(" ");
Thread p1 = new Thread(new Produttore1("p1", 1, q, text));
Thread c1 = new Thread(new Consumatore1("c1", 2, q));
Thread c2 = new Thread(new Consumatore1("c2", 2, q));
p1.start(); c1.start(); c2.start();
Thread.sleep(20 * 1000);
p1.interrupt(); c1.interrupt(); c2.interrupt();
System.out.println(q);}}
```

p1 Nel mezzo del cammin di nostra vita mi ritrovai per una selva oscura Nel mezzo del cammin di nostra vita mi ritrovai per una selva oscura Nel mezzo del cammin di nostra vita mi ritrovai per una selva oscura Nel

c1 Nel del nostra ritrovai per selva Nel del di mi per selva mezzo del cammin nostra mi per una selva **[oscura, Nel]**

c2 mezzo cammin di vita mi una oscura mezzo cammin nostra vita ritrovai una oscura Nel di vita ritrovai

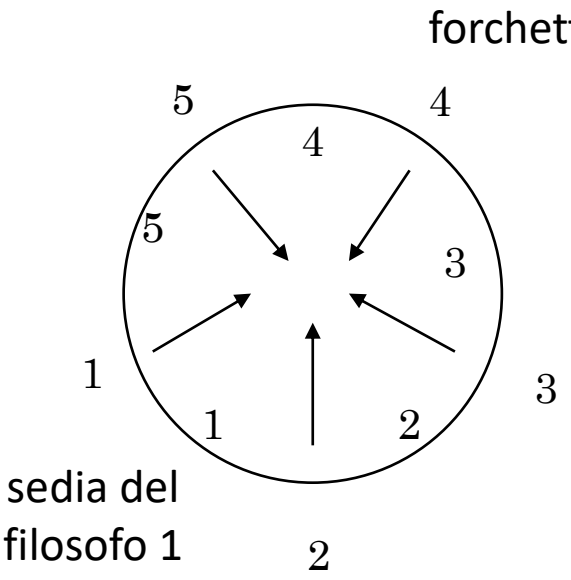
Esempio 5 filosofi

I filosofi ripetutamente lavorano e poi si recano in sala da pranzo per mangiare. La tavola ha 5 forchette e ciascuno ne usa 2.

La tavola registra il n. di attese (nWait) e il n. di successi (n) per filosofo.

Il periodo in cui un filosofo pensa (mangia) dura fino a 2 (3) secondi.

Per evitare un deadlock ciascun filosofo deve prendere simultaneamente le 2 forchette e non una per volta.



Nota: nell'implementazione filosofi e forchette sono numerati da 0 a 4.

Tavola

```
import java.util.Arrays;
public class Tavola {
        private int[] nWait = new int[5]; private int[] n = new int[5];
        private boolean[] forchette = {true, true, true, true, true};
public synchronized void prendiForchette (int id) throws InterruptedException{
        while (!forchette[id] || !forchette[(id+1)%5]) {nWait[id]++; wait();}
        forchette[id] = false; forchette[(id+1)%5] = false; n[id]++;
        notifyAll();
}
public synchronized void rendiForchette (int id)  {
        forchette[id] = true; forchette[(id+1)%5] = true;
        notifyAll();
}
public synchronized String toString() {
        return Arrays.toString(nWait) + " " + Arrays.toString(n);
}}
```

nWait = n. di wait per filosofo; n = n. di successi per filosofo.

Filosofo

```
public class Filosofo implements Runnable{
        private int id; private Tavola tavola;
        public Filosofo (int id, Tavola tavola) {this.id = id; this.tavola = tavola;}
        public void run() {
                int intervallo1 = 2; int intervallo2 = 3;
                long p1 = (long)(Math.random() * 1000 * intervallo1);
                long p2 = (long)(Math.random() * 1000 * intervallo2);
                try {
                while (true){
                        Thread.sleep(p1);
                        tavola.prendiForchette(id);
                        System.out.println(id + " inizia il pranzo");
                        Thread.sleep(p2);
                        System.out.println(id + " finisce il pranzo");
                        tavola.rendiForchette(id);
                }
                } catch (InterruptedException e) {}
}
```

Il periodo in cui un filosofo pensa (mangia) dura fino a 2 (3) secondi

Nota: p1 e p2 si potrebbero spostare nel while.

Main

```
public static void main(String[] args) throws InterruptedException {
Tavola tavola = new Tavola();
Thread[] filosofi = new Thread[5];
for (int i = 0; i < 5; i++) filosofi[i] = new Thread(new Filosofo(i, tavola));
for (int i = 0; i < 5; i++) filosofi[i].start();
Thread.sleep(10 * 1000);
for (int i = 0; i < 5; i++) filosofi[i].interrupt();
System.out.println(tavola);
}
```

```
0 inizia il pranzo
3 inizia il pranzo
0 finisce il pranzo
1 inizia il pranzo
3 finisce il pranzo
4 inizia il pranzo
4 finisce il pranzo
1 finisce il pranzo
2 inizia il pranzo
0 inizia il pranzo
2 finisce il pranzo
3 inizia il pranzo
0 finisce il pranzo…
[4, 3, 6, 0, 11] [4, 3, 3, 3, 3]
```

Patterns

Patterns

Reusable solutions to recurring problems (Christopher Alexander)

Software patterns were motivated by Christopher Alexander's work on patterns in architecture.

A PATTERN LANGUAGE, Oxford University Press, New York, 1977.

"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice".

The book presents 253 patterns for regions, towns, houses, gardens and rooms.

Design patterns

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides ("Gang of Four").

Design Patterns: Elements of Reusable Object-Oriented Software.

Addison-Wesley, 1994.

"It's a book of design patterns that describes simple and elegant solutions to specific problems in object-oriented software design. Design patterns capture solutions that have developed and evolved over time. ..

Each design pattern systematically names, explains, and evaluates an important and recurring design in object-oriented systems. Our goal is to capture design experience in a form that people can use effectively."

Creational patterns: factory method, builder, prototype, singleton …

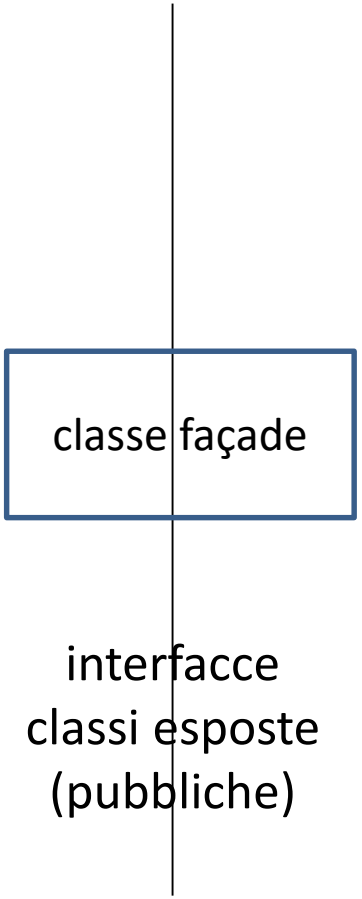
Structural patterns: façade, composite, decorator …

Behavioral patterns: iterator, command, observer …

Façade

package di uso
con classi di uso (o di test
o interfacce grafiche)

Le classi di uso agiscono in modo controllato sugli oggetti applicativi.



package applicativo
con classi applicative

Gli oggetti delle classi applicative sono generati dai metodi (factory) della classe principale (facciata). La classe principale contiene le collezioni degli oggetti applicativi.

classi visibili nel package

Factory method

Obiettivo: generare un nuovo oggetto senza chiamare direttamente un costruttore.

Inserimenti
La biblioteca può avere più copie (volumi) dello stesso libro.
Il metodo **addLibro** (titolo, nVolumi, autori) inserisce un libro e i volumi corrispondenti.

Il metodo **addUtente** (nome, maxPrestiti, durata) inserisce un utente con il nome, il n. max di prestiti che può avere nello stesso periodo, la durata (massima) in giorni dei suoi prestiti.

Builder

Intent: to separate the construction of a complex object from its final representation; e.g. StringBuilder and String.

Example
A car has a number of options, such as the number of seats, the presence of a trip computer/a bluetooth kit.

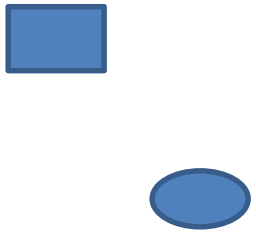
Class CarBuilder enables you to configure a car by choosing the options.
When you have finished, you get the configuration of the car with the price.

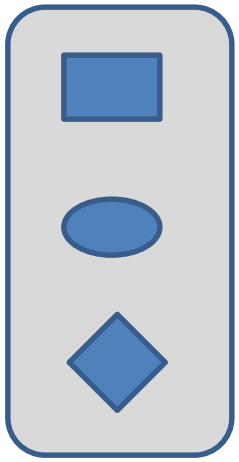
```
CarBuilder carBuilder = new CarBuilder();
carBuilder.setNofSeats(5); carBuilder.setTripComputer();
Car car = carBuilder.getResult();
```

This pattern is frequently used to build products made up of components.

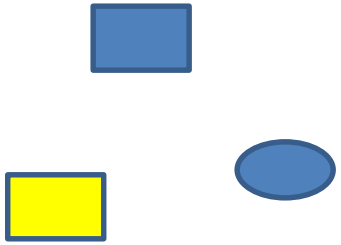
Prototype

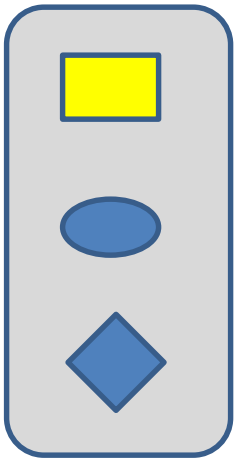
Intent: to get a new object by making a copy of a prototypical instance.
The class of the prototype implements the clone method.





A diagram is made up of objects similar to those included in a palette.
The objects in the palette are prototypes. If the user modifies the properties of a prototype the new instances will be different from the previous ones.





Singleton pattern

Garantisce che di una classe esista un'unica istanza.

Serve, ad esempio, per la gestione di dispositivi.

Esempio

```
public class Dispositivo {
private Dispositivo() {} // nota: costruttore privato
private static Dispositivo dispositivo = new Dispositivo();
public static Dispositivo getDispositivo() {return dispositivo;}
private boolean stato = true;
// start se lo stato è true, stop se false
public void start() {
        if(stato) {System.out.println("start"); stato = false;}
}
public void stop() {
        if(!stato){System.out.println("stop"); stato = true;}
}
}
```

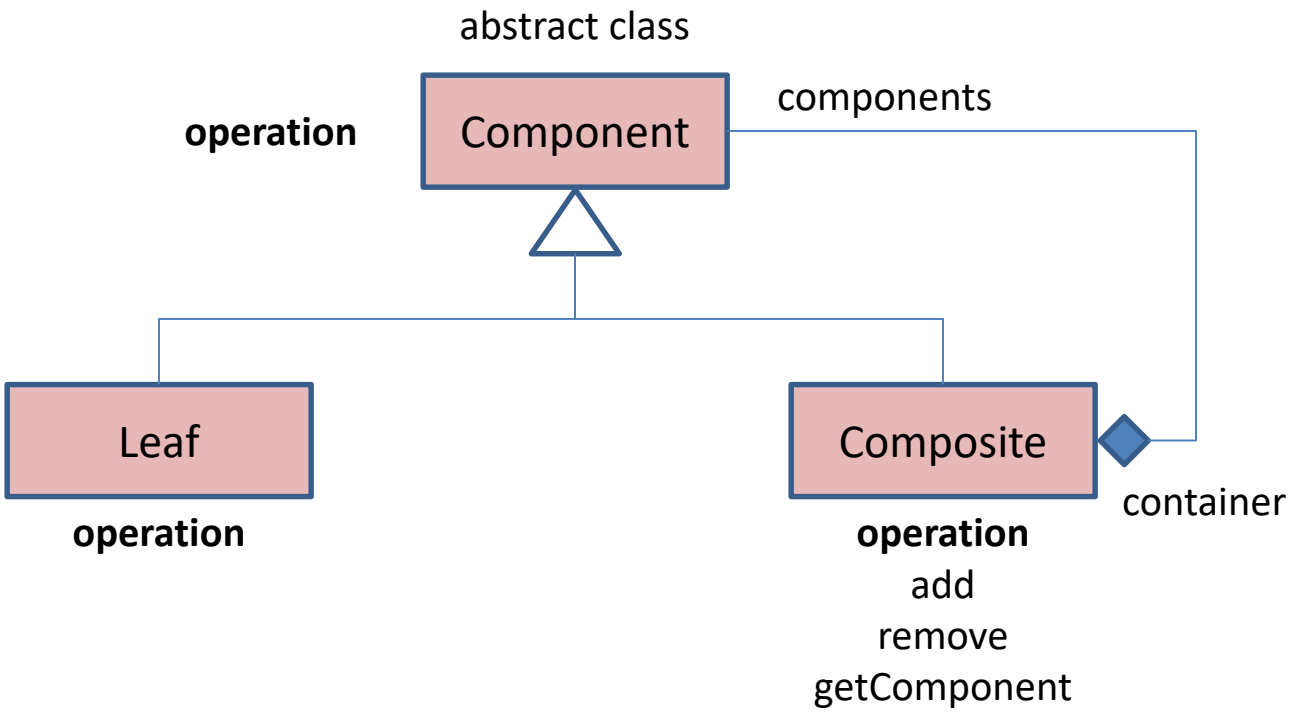
Esempio

```
public static void main(String[] args) {
Dispositivo d = Dispositivo.getDispositivo();
d.start(); d.start(); d.stop(); d.stop();
d.start(); d.stop();
}
```

```
start
stop
start
stop
```

Composite

Intent: to handle part-whole hierarchies by treating simple objects and compound ones uniformly.



Iterator

Intent: to provide a way to access the elements of a collection sequentially without exposing the underlying representation.

Implemented by Iterator and ListIterator objects.

Command

Intent: to encapsulate the request for an operation into an object.

Example
A client generates a command and passes it to an invoker; the invoker sets the command performer and passes it the command.
In alternative, the performer may be set by the client.
In addition, the client may be notified with the result.

Observer

Intent: to enable a number of observers to be notified when a subject changes state.
Also known as *publish/subscribe*.

The subject provides an interface to add/remove observers.
The subject knows the observers through an interface containing the notify method.

In java swing observers are called *event listeners*.

Esempio d'uso del pattern publish/subscribe

Ci sono agenzie d'informazione e utenti. Le agenzie pubblicano notizie di un certo tipo e le inviano ai loro iscritti (utenti). Per ricevere notizie gli utenti si devono iscrivere presso le agenzie.

Le notizie sono oggetti di classe **Notizia** o derivate come **NotiziaEconomica** e **NotiziaSportiva**.

Agenzia è una classe generica che implementa l'interfaccia AgenziaI

```
public class Agenzia <E extends Notizia> implements AgenziaI
```

L'interfaccia consente l'iscrizione di un utente presso un'agenzia.

```
public interface AgenziaI {
        void addUtente(UtenteI utente);
}
```

Esempio d'uso del pattern publish/subscribe

Gli utenti implementano l'interfaccia UtenteI che consente loro di iscriversi presso le agenzie a cui sono interessati e di ricevere le notizie.

```
public interface UtenteI {
        void iscriviti (Agenzia <? extends Notizia> agenzia);
        void riceviNotizia (Notizia notizia);
}
```

main di esempio

```
Agenzia<Notizia> agenziaN = new Agenzia<>("agenziaN");
Agenzia<NotiziaEconomica> agenziaE = new Agenzia<>("agenziaE");
Agenzia<NotiziaSportiva> agenziaS = new Agenzia<>("agenziaS");

Utente john = new Utente("john"); Utente mary = new Utente("mary");
john.iscriviti(agenziaN); john.iscriviti(agenziaS);
mary.iscriviti(agenziaN); mary.iscriviti(agenziaE);

agenziaN.pubblica(new Notizia("Aumento dei turisti in Italia"));
agenziaE.pubblica(new NotiziaEconomica("Borse fluttuanti"));
agenziaS.pubblica(new NotiziaSportiva("Presentato il nuovo Tour"));
john.leggi(); mary.leggi();
```

```
john legge: Notizia: Aumento dei turisti in Italia
john legge: NotiziaSportiva: Presentato il nuovo Tour
mary legge: Notizia: Aumento dei turisti in Italia
mary legge: NotiziaEconomica: Borse fluttuanti
```

Soluzione

```
public class Notizia {
private String contenuto;
public Notizia (String contenuto) {this.contenuto = contenuto;}
public String toString() {
        return this.getClass().getSimpleName() + ": " + contenuto;}
}

public class NotiziaEconomica extends Notizia {
public NotiziaEconomica (String contenuto) {super(contenuto);}
}

public class NotiziaSportiva extends Notizia {
public NotiziaSportiva (String contenuto) {super(contenuto);}
}
```

Soluzione

```
public class Agenzia <E extends Notizia> implements AgenziaI {
private String nome; public String getNome() {return nome;}
private List<UtenteI> utenti = new ArrayList<>();
public void addUtente(UtenteI utente) {
        utenti.add(utente);
}
public Agenzia (String nome) {this.nome = nome;}
public void pubblica (E notizia) {
        for (UtenteI u: utenti) u.riceviNotizia(notizia);}
}
```

Soluzione

```
public class Utente implements UtenteI {
private String nome; public String getNome() {return nome;}
public Utente (String nome) {this.nome = nome;}
public void iscriviti (Agenzia <? extends Notizia> agenzia) {
        agenzia.addUtente(this);
}
private List<Notizia> notizie = new ArrayList<>();
public void riceviNotizia (Notizia notizia) {notizie.add(notizia);}
public void leggi() {for(Notizia notizia: notizie)
        System.out.println(nome + " legge: " + notizia); notizie.clear();}
}
```

Livelli dei pattern

Da specifici a generici

programmazione

come risolvere un particolare problema; ad es. leggere tutto un file testuale linea dopo linea.

progetto (design)

schema di classi e interfacce per ottenere un certo comportamento; ad es. porre in relazione fonti di notizie con entità interessate a riceverle.

architettura

schema di sottosistemi e intercomunicazioni per ottenere un certo comportamento; ad es. il pattern MVC (Model View Controller) per applicazioni con interfacce grafiche.

Interfacce grafiche

Programmazione ad eventi

Introduzione a swing

Introduzione a javafx

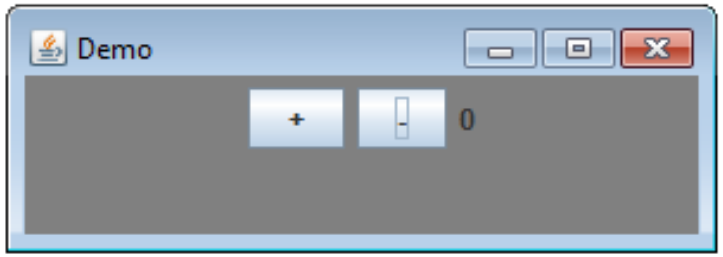
Swing

Un'interfaccia è costituita da un contenitore principale, eventuali contenitori intermedi ed elementi.

Classi dei contenitori principali: **JFrame**, JDialog, JApplet.

Alcune classi di contenitori intermedi: **JPanel**, JScrollPane.

Alcune classi di elementi (controlli): **JButton**, **JLabel**, **JTextField**, **JTextArea**, JList.

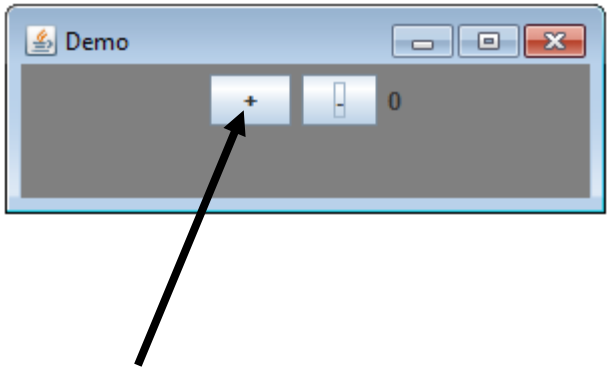


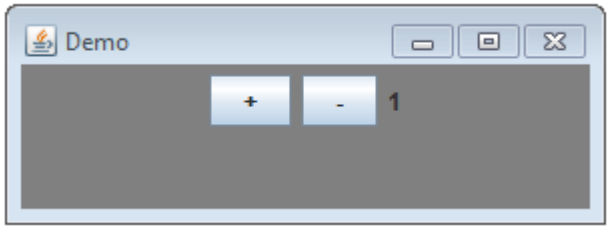
Nota: l'interfaccia si definisce solitamente con una sottoclasse di una classe che definisce un contenitore principale.

Swing

La disposizione dei componenti è organizzata mediante un layout manager che può essere di vario tipo. I principali sono: **FlowLayout** (predefinito per JPanel), **BorderLayout** (predefinito per i contenitori principali), BoxLayout, GridLayout, GridBagLayout.

L'utente impartisce comandi all'applicazione mediante i controlli attivi; essi emettono eventi indirizzati agli ascoltatori (listeners) registrati.





Con un click sul pulsante + si incrementa un contatore il cui valore è mostrato nella label a destra.

Eventi

L'ascoltatore di un JButton deve implementare l'interfaccia

ActionListener (in java.awt.event)

che contiene l'unico metodo

void **actionPerformed**(ActionEvent e)

Della classe ActionEvent (in java.awt.event) si usa principalmente il metodo

public String **getActionCommand**()

che serve ad identificare il pulsante che ha emesso l'evento.

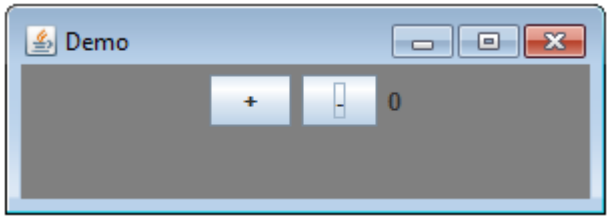
L'ascoltatore (listener) può essere l'oggetto dell'interfaccia oppure un oggetto separato (interno, esterno, lambda expression).

Nota: ActionListener è un'interfaccia funzionale.

Esempio

Per implementare l'esempio si scrive la classe ButtonDemo che eredita da JFrame. La classe riceve gli eventi emessi dai pulsanti quindi implementa ActionListener.

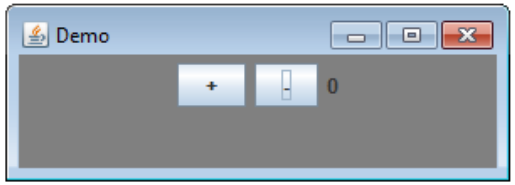
La struttura si definisce nel costruttore: si usa un FlowLayout per avere i tre controlli allineati.



Esempio

```
import javax.swing.*;
import java.awt.*;import java.awt.event.*;
@SuppressWarnings("serial")
public class ButtonDemo extends JFrame implements ActionListener {
   public static final int WIDTH = 300; public static final int HEIGHT = 100;
   int val = 0; JLabel labelV; // val è il contatore
   public ButtonDemo()
   {  setSize(WIDTH, HEIGHT); // è un metodo di awt.Component
      setTitle("Demo");  // è un metodo di awt.Frame
      Container contentPane = getContentPane(); // Container  è una classe di awt
      contentPane.setBackground(Color.GRAY);  // anche Color

      contentPane.setLayout(new FlowLayout());
      JButton buttonP = new JButton("+");
      buttonP.addActionListener (this); contentPane.add(buttonP);
      JButton buttonM = new JButton("-");
      buttonM.addActionListener(this); contentPane.add(buttonM);
      labelV = new JLabel("0"); contentPane.add(labelV); }
```



Esempio

```
public void actionPerformed (ActionEvent e)
{
      if (e.getActionCommand().equals("+"))
            labelV.setText(String.valueOf(++val));
      else if (e.getActionCommand().equals("-"))
            labelV.setText(String.valueOf(--val));
   }

public static void main (String[] args) {
SwingUtilities.invokeLater(new Runnable() {
// per attivare l'applicazione dall'event-dispatching thread
    public void run() {
            ButtonDemo gui = new ButtonDemo();
            gui.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            gui.setVisible(true);
     }
     });
}}
```

Nota
Gli oggetti grafici che devono essere accessibili al metodo actionPerformed vanno dichiarati come attributi d'istanza.

Listener esterno

```
public class ButtonDemo1 extends JFrame {
   public static final int WIDTH = 300;  public static final int HEIGHT = 100;
   int val = 0; JLabel labelV;
public ButtonDemo1(){setSize(WIDTH, HEIGHT); setTitle("Demo");
          Container contentPane = getContentPane();
          contentPane.setBackground(Color.GRAY);
          contentPane.setLayout(new FlowLayout());
          Listener l = new Listener();
          JButton buttonP = new JButton("+");buttonP.addActionListener(l);
          contentPane.add(buttonP);
          JButton buttonM = new JButton("-");buttonM.addActionListener(l);
          contentPane.add(buttonM);
          labelV = new JLabel("0"); contentPane.add(labelV);}
private class Listener implements ActionListener {
public void actionPerformed(ActionEvent e)
   {
     if (e.getActionCommand().equals("+"))
              labelV.setText(String.valueOf(++val));
     else if (e.getActionCommand().equals("-"))
              labelV.setText(String.valueOf(--val));
   }
}}
```

Con lambda expression

```
public class ButtonDemoLambda extends JFrame {
public static final int WIDTH = 300; public static final int HEIGHT = 100;
int val = 0; JLabel labelV;
public ButtonDemoLambda()
{setSize(WIDTH, HEIGHT); setTitle("Demo");
Container contentPane = getContentPane();
contentPane.setBackground(Color.GRAY);
contentPane.setLayout(new FlowLayout());
JButton buttonP = new JButton("+");
buttonP.addActionListener (e ->   {labelV.setText(String.valueOf(++val));});
contentPane.add(buttonP);
JButton buttonM = new JButton("-");
buttonM.addActionListener (e -> {labelV.setText(String.valueOf(--val));});
contentPane.add(buttonM);
labelV = new JLabel("0"); contentPane.add(labelV); }
```

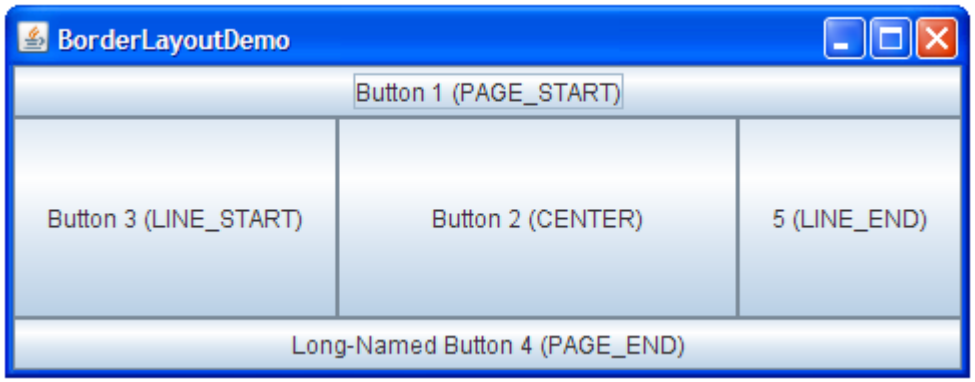
NOTA:

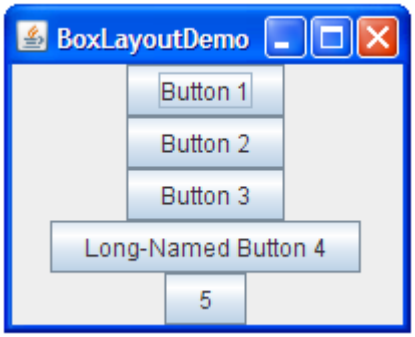
Le lambda expression mostrate hanno side effects.

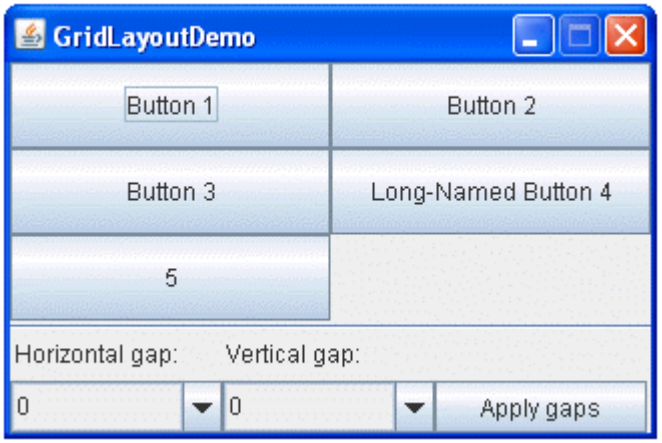
Gerarchia di classi grafiche

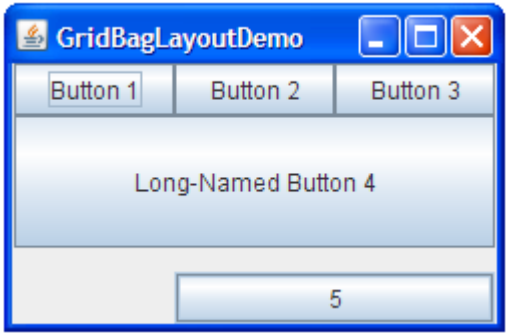
java.lang.Object
java.awt.Component
java.awt.Container
java.awt.Window
java.awt.Frame
javax.swing.JFrame

Layout manager









da http://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html

Alcune interfacce per ricevere eventi

ActionListener: da pulsanti, menu, return in casella di testo.

WindowListener: da pulsanti di gestione della finestra (chiusura, riduzione a icona, ingrandimento).

ListSelectionListener: da scelta in lista.

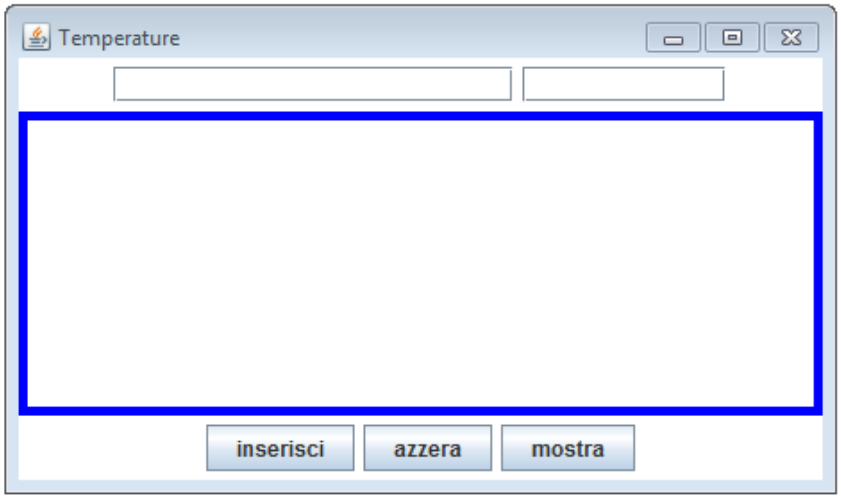
MouseListener: da comandi del mouse.

Esempio con MVC (Model View Controller)

Si progetti un'interfaccia grafica che consenta l'inserimento di una serie di temperature per varie città; le temperature sono valori interi. Il comando *azzera* cancella gli elementi già inseriti; il comando *mostra* visualizza l'elenco ordinato per città.

JTextField **cittàTF**; JTextField **temperaturaTF**;

private JTextArea **areaTA**



La struttura è basata su 3 pannelli collocati lungo l'asse verticale del BorderLayout del contenitore.
Il listener è un oggetto di classe Controllore; i dati sono contenuti nella classe Modello.

Esempio

```
public class Temperature extends JFrame {
private static final int LINES = 10;
public static final int CHAR_PER_LINE = 40;
private JTextField cittàTF; private JTextField temperaturaTF;
private JTextArea areaTA; private Modello modello = new Modello();
private ActionListener controllore = new Controllore();

// aggiunge n pulsanti al pannello
private void addButtons (JPanel buttonPanel, String... labels){
        for (String label: labels) {JButton button = new JButton(label);
                button.addActionListener(controllore);
                buttonPanel.add(button);}}
// aggiunge un TextField al pannello e ne dà il rif. che serve come attributo.
private JTextField addTF (String label, int length, JPanel panel){
        JTextField tf = new JTextField(label,length);
        tf.setBackground(Color.white); panel.add(tf); return tf;}
```

Esempio

```
public Temperature () {
setTitle("Temperature");
Container contentPane = getContentPane();
        JPanel buttonPanel = new JPanel();
        buttonPanel.setBackground(Color.white);
        addButtons(buttonPanel,"inserisci", "azzera", "mostra");
        contentPane.add(buttonPanel, BorderLayout.SOUTH);
JPanel areaPanel = new JPanel();  areaPanel.setBackground(Color.blue);
areaTA = new JTextArea(LINES, CHAR_PER_LINE);
areaTA.setBackground(Color.white); areaPanel.add(areaTA);
contentPane.add(areaPanel, BorderLayout.CENTER);
        JPanel textPanel = new JPanel();
        textPanel.setBackground(Color.white);
        cittàTF = addTF("", 20, textPanel);
        temperaturaTF = addTF("", 10, textPanel);
        contentPane.add(textPanel, BorderLayout.NORTH); }
```

Esempio

```
static class Modello {
private Map<String, Integer> mappa = new TreeMap<String, Integer>();
void inserisci (String città, int t) {mappa.put(città,  t);}
void azzera () {mappa.clear();}
public String toString() {return mappa.toString();}
}
```

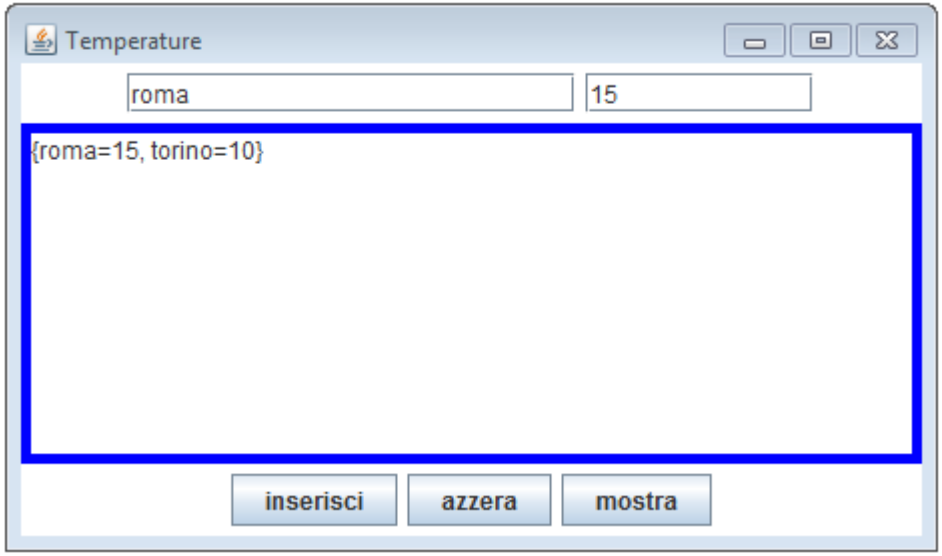
Esempio

```
class Controllore implements ActionListener {
public void actionPerformed (ActionEvent e) {
areaTA.setText("");
String actionCommand = e.getActionCommand();
if (actionCommand.equals("inserisci")){
        String città = cittàTF.getText();
        if (!città.equals("")){
        try{
                modello.inserisci(città,
                        Integer.valueOf(temperaturaTF.getText()));
                areaTA.setText("inserite");
        }catch(NumberFormatException ex){}
        }
} else if (actionCommand.equals("azzera"))
        modello.azzera();
else if (actionCommand.equals("mostra")){
        areaTA.setText(modello.toString());}} }
```

Esempio

```
public static void main(String[] args) {
SwingUtilities.invokeLater(new Runnable() {
    public void run() {
            Temperature gui = new Temperature();
            JFrame.setDefaultLookAndFeelDecorated(true);
            gui.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            gui.pack(); gui.setVisible(true);}
    });
}}
```

invokeLater accoda l'oggetto per l'event dispatching thread



JavaFX

Toolkit per scrivere rich client applications in java.

3 aspetti fondamentali:
grafica; può essere definita separatamente in file css
struttura; può essere definita separatamente in file xml
logica; basata su eventi, osservatori (listeners), proprietà

Struttura


```
public class Contatore extends Application {
public void start (Stage stage) {

BorderPane pane = new BorderPane();
Scene scene = new Scene(pane);
stage.setScene(scene);
stage.setTitle("contatore");
stage.setMinHeight(100);
stage.setMinWidth(300);
stage.show();
}
```

Un'interfaccia si programma in una classe derivata da Application.
Application contiene il metodo abstract **start**, il quale riceve il top level container di tipo Stage.
Lo stage stabilisce la scena corrente. La scena contiene gli elementi dell'interfaccia .
Gli elementi sono strutturati in gruppi o in layout.

Layout

Alcuni layout: BorderPane, HBox, VBox, GridPane

Esempio

```
HBox box = new HBox(10); // 10 pixel di spazio tra i nodi contenuti
box.setAlignment(Pos.CENTER); // allineamento dei componenti (centrati)
BorderPane pane = new BorderPane();
pane.setCenter(box); // il box nella zona centrale
```

Tre controlli sono collocati nel box.

Controlli

Alcuni controlli: Label, Button, Text Field, List View.


```
Label valLb = new Label("value: 0");
valLb.setTextFill(Color.GREEN);
valLb.setPrefWidth(100);

Button incr = new Button("incr");
Button decr = new Button("decr");
HBox box = new HBox(10);
        box.getChildren().addAll (incr,decr,valLb);
        box.setAlignment(Pos.CENTER);
        box.setPadding(new Insets(10)); // distanza dai bordi del pannello
```

Azioni associate ai pulsanti

Si stabiliscono con setOnAction.

public final void **setOnAction**(EventHandler<ActionEvent> value)

L'interfaccia EventHandler<T extends Event> è funzionale e contiene il metodo astratto

void handle(T event).

Un evento ha associati: event source, event target, event type.

Spesso l'event handler si definisce con una lamba expression.

Esempio:

```
incr.setOnAction(e -> valLb.setText("value: "+ ++val + ""));
decr.setOnAction(e -> valLb.setText("value: "+ --val + ""));
```

Posizione di Button nell'albero delle classi

java.lang.Object
- javafx.scene.Node
  - javafx.scene.Parent
    - javafx.scene.layout.Region
      - javafx.scene.control.Control
        - javafx.scene.control.Labeled
          - javafx.scene.control.ButtonBase
            - javafx.scene.control.Button

Si notino i vari packages: scene, scene.layout, scene.control.

Class Contatore

```
package application;
import javafx.application.Application;
import javafx.geometry.*;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.*;
import javafx.scene.paint.Color;


public class Contatore extends Application {
int val = 0;
Label valLb = new Label("value: 0");
```

Class Contatore


```
public void start(Stage stage) {
valLb.setTextFill(Color.GREEN);
valLb.setPrefWidth(100);

Button incr = new Button("incr");
Button decr = new Button("decr");
incr.setOnAction(e -> valLb.setText("value: "+ ++val + ""));
decr.setOnAction(e -> valLb.setText("value: "+ --val + ""));

HBox box = new HBox(10);
box.getChildren().addAll(incr,decr,valLb);
box.setAlignment(Pos.CENTER);
box.setPadding(new Insets(10));

BorderPane pane = new BorderPane();
pane.setCenter(box);
Scene scene = new Scene(pane);
```

Class Contatore

```
String cssFile = "file:contatore.css";
scene.getStylesheets().add(cssFile);

stage.setScene(scene);
stage.setTitle("contatore");
stage.setMinHeight(100);
stage.setMinWidth(300);
stage.show();
}

public static void main(String[] args) {
launch(args);
}
}
```

File Contatore.css

```
.root{
    -fx-font-size: 14pt;
    -fx-font-family: "Tahoma";
 }
 .button{
    -fx-cell-size: 200;
    -fx-text-fill: #006464;
    -fx-padding: 5;
    -fx-effect: dropshadow( one-pass-box , black , 8 , 0.0 , 2 , 0 );
}
```

Gli stili possono agire in generale, per tipo di elemento o sul singolo elemento.
Si possono definire anche nel programma;
ad es. valLb.setTextFill(Color.*GREEN);*

https://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html