Politecnico di Milano
Facoltà di Ingegneria dell'Informazione
Dipartimento di Elettronica e Informazione

Esercitazioni di Ingegneria del Software

05: Swing & UML

Esercizio: cosa stampa?

```
class Father {
  private int x;

  public Father(int x) { this.x = x; }

  public int m(Father f) {
    return (f.x - this.x);
  } }

class Son extends Father {
  private int y;

  public Son(int x, int y) {super(x); this.y = y; }

  public int m(Father f) { return 100; }

  public int m(Son s) {
    return super.m(s) + (s.y - this.y);
  }}
```

Esercizio: cosa stampa?

```
public class Launcher {
public static void main(String args[]) {
  Father f1, f2;    Son s1, s2;
  f1 = new Father(3);
  f2 = new Son(3,10);
  System.out.println(f1.m(f2));              /* 1 */
  System.out.println(f2.m(f1));              /* 2 */
  s1 = new Son(4,21);
  System.out.println(s1.m(f1) + s1.m(f2));   /* 3 */
  System.out.println(f1.m(s1) + f2.m(s1));   /* 4 */
  s2 = new Son(5,22);
  System.out.println(s1.m(s2));              /* 5 */
}
}
```


Esercizio: cosa stampa?

- Risposta: La classe Son definisce un metodo m(Father), che effettua un overriding del metodo m(Father) in Father, e un overloading di m, con signature m(Son).
- Istruzione 1:
  - Parte statica (overloading): f1 ha tipo statico Father -> il metodo viene cercato nella classe father. f2 ha tipo statico Father -> viene cercato un metodo la cui signature è compatibile con m(Father). Il metodo viene trovato, e' proprio Father.m(Father), e occorre cercare tra i metodi che ne effettuano un overriding.
  - Parte dinamica (overriding): f1 ha tipo dinamico Father -> viene scelto il metodo Father.m(Father). Stampato f2.x - f1.x, ossia 0.
- Istruzione 2:
  - Parte statica (overloading): ancora, f1 e f2 hanno tipo statico Father. Quindi viene sempre scelto Father.m(Father) (o uno che ne fa overriding).
  - Parte dinamica (overriding): stavolta f2 ha tipo dinamico Son, e quindi viene scelto il metodo Son.m(Father), che effettua overriding. Viene stampato 100.

Esercizio: cosa stampa?

- Istruzione 3:
  - Parte statica: le due chiamate hanno come tipo statico Son.m(Father). Quindi viene scelto questo metodo, o un metodo che ne fa override...
  - Parte dinamica: ...ma nessun metodo fa override di Son.m(Father), quindi per entrambe le chiamate viene eseguito questo. Notare che, nonostante f2 abbia tipo dinamico Son, s.m(f2) NON esegue Son.m(Son)!!! Viene stampato 200.
- Istruzione 4:
  - Parte statica: le due chiamate hanno come tipo statico Father.m(Son). Non esiste un metodo con questa signature, ma Father.m(Father) è compatibile. Viene scelto quindi Father.m(Father), o un metodo che ne fa override.
  - Parte dinamica: dal momento che f1 e f2 hanno diversi tipi dinamici, la prima chiamata usa il metodo Father.m(Father), la seconda usa il metodo overridden Son.m(Father). Il risultato è 1 + 100 = 101.

Esercizio: cosa stampa?

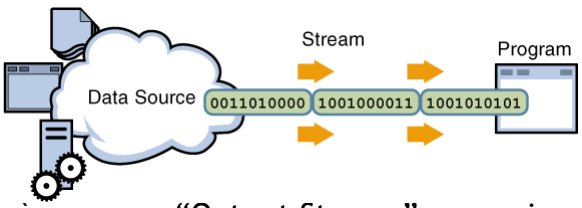
- Istruzione 5:
  - Statico è Son.m(Son), e non ci sono metodi che ne fanno overriding. All'interno, viene effettuata una chiamata di super.m(s), con s parametro il cui tipo statico è Son; super significa "della superclasse statica" - quindi di Father. Staticamente, questo significa cercare Father.m(Son), che non esiste: Però esiste Father.m(Father), che è compatibile. A runtime viene invocato questo. Quindi, super.m(s) ritorna 1, e l'istruzione 5 stampa 2 a schermo.

Java I/O

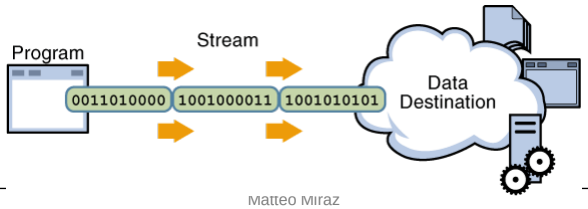
- Idea di base:
  - Un flusso (Stream) rappresenta una sorgente di ingresso (o di uscita).
  - Rappresentati dalle classe astratte
    - InputStream → int read()
    - OutputStream → write(int value)
  - Può rappresentare diversi tipi di sorgenti e di destinazione
    - Es. File su disco, risorse http, array condivisi, …
  - È possibile effettuare diverse operazioni sui flussi:
    - Alcuni trasferiscono solamente i dati (magari facendo del buffering)
    - Altri flussi permettono di modificare il flusso (es. (de)comprimendolo)

Java I/O

- I flussi rappresentano una sequenza di dati
  - Si può usare un "Input Stream" per leggere dati



  - Si può usare un "Output Stream" per scrivere i dati



Java I/O: usare i File

- Facciamo la copia da un file ad un altro

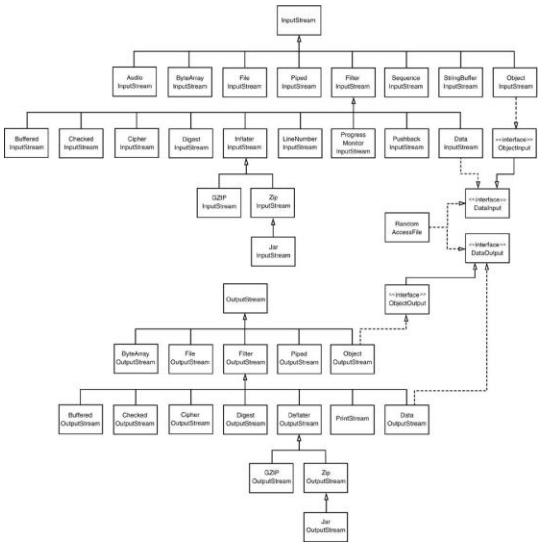
```
public void copia(String src, String dest)
      throws IOException {
  InputStream in = new FileInputStream(src);
  OutputStream out = new FileOutputStream(dest);

  int c;
  while ((c = in.read()) != -1)
      out.write(c);

  in.close();
  out.close();
}
```


Java I/O: Flessibilità

- Scarichiamo un file:
  - È una variante della copia!!!

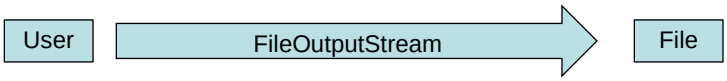
```
public void download(String dest)
      throws IOException {

  URL u = new URL("http://dei.polimi.it/miraz/index.html");
  InputStream in = u.openStream();

  FileOutputStream out = new FileOutputStream(dest);

  int c;
  while ((c = in.read()) != -1) out.write(c);

  in.close();
  out.close();
}
```

Java I/O: Flessibilità

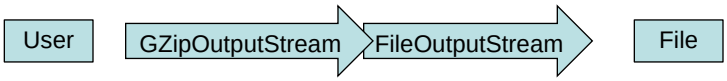


Java I/O: Componibilità

Salvare su un file



Salvare su un file compresso



Salvare su un file compresso e cifrato

Java I/O: Componibilità

- Creare un offset output stream, che trasli i valori da scrivere

```
class OffsetStream extends OutputStream {
  private OutputStream out;
  private int offset;

  public OffsetStream(OutputStream out, int off) {
      this.out = out; this.offset = off;
  }

  public void write(int b) throws IOException {
      out.write((b + offset) % 255);
  }
}
```


Java I/O: Flessibilità

- Esercizio:
  - Creare un'output stream in grado di scrivere i dati su un file, in maniera efficente, e comprimendoli

```
OutputStream o =
  new GZIPOutputStream(
      new BufferedOutputStream(
          new FileOutputStream(fileName)));
```

Java I/O: Caratteri

- Per gestire i flussi di caratteri, possiamo usare
  - Reader per leggere
    - Le sottoclassi gestiscono casi particolari
      - Per leggere una riga intera, usare `BufferedReader`
  - Writer per scrivere
    - Le sottoclassi gestiscono casi particolari
      - Per scrivere una riga intera, usare `PrintWriter`

- Problema:
  - Flussi e Reader/Writer sono compatibili?
    - NO
    - È necessario usare convertitori apposta
      - `Reader ir = new InputStreamReader(inputStream);`
      - `Writer ow = new OutputStreamWriter(outputStr);`

Java I/O: Caratteri

```
public void copiaTxt(String src, String dest)
        throws IOException {

   InputStream inputStream = new FileInputStream(src);
   InputStreamReader ir = new InputStreamReader(inputStream);
   BufferedReader reader = new BufferedReader(ir);

   OutputStream outputStream = new FileOutputStream(dest);
   OutputStreamWriter ow = new OutputStreamWriter(outputStream);
   PrintWriter writer = new PrintWriter(ow);

   String l;
   while ((l = reader.readLine()) != null)
        writer.println(l);

   reader.close();
   writer.close();
}
```

Java I/O: e la tastiera?

java.lang
Class System

java.lang.Object
  └ java.lang.System

Field Detail

in

```
public static final InputStream in
```

The "standard" input stream. This stream is already open and ready to supply input data. Typically this stream corresponds to keyboard input or another input source specified by the host environment or user.

- Viene gestita come un normale flusso di ingresso
  - System.in contiene il flusso di dati provenienti dalla tastiera
  - È un flusso di caratteri?
    - In genere sì…
    - … ma chi è abituato all'ambiente *nix sa che è possibile concatenare comandi:

      cat filebinario.png| png2pdf > immagine.pdf

Java I/O: e la tastiera?

```
public void echoDaTastiera() throws IOException {

   BufferedReader read = new BufferedReader(
       new InputStreamReader(System.in));

   String l;
   while((l = read.readLine()) != null)
       System.out.println(">" + l);
}
```

Java I/O: e la tastiera?

La classe Scanner:

"A simple text scanner which can parse primitive types and strings using regular expressions."

```
public void calcolaMedia() {
   Scanner sc = new Scanner(System.in);

   System.out.println("Quanti elementi?");
   int num = sc.nextInt();

   float tot = 0;
   for (int i = 0; i < num; i++) {
        float val = sc.nextFloat();
        tot += val;
   }

   System.out.println("media: " + tot / num);
}
```


Classi anonime

- Molte volte capita di dover creare una sola istanza di una classe …
  - Esempio tipico: reagire ad un evento della GUI

```
JButton ok = new Jbutton("OK");
ok.addActionListener(new BottoneOKActionListener());

public class BottoneOKActionListener
  implements ActionListener {
  public void actionPerformed(ActionEvent arg0) {
    System.out.println("OK");
  }
}
```

Classi anonime

- È possibile creare classi anonime:

```
JButton ok = new JButton("OK");
ok.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent arg0) {
    System.out.println("OK");
  }
});
```


Simple GUI

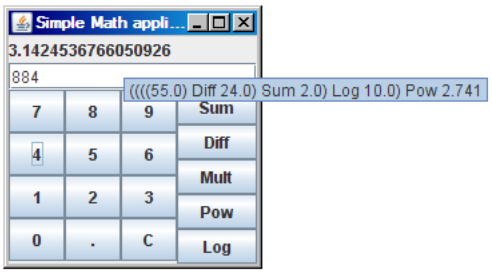
```
public class Launcher {
  public static void main(String[] args) {
    GUI g = new GUI();
    g.setVisible(true);
} }

class GUI extends JFrame {
  private JLabel label;

  public GUI() {
    setLayout(new BorderLayout());

    label = new JLabel("Cliccami! (0)");
    this.getContentPane().add(label, BorderLayout.CENTER);
    label.addMouseListener(new MouseListener() {

      private int n = 0;

      public void mouseClicked(MouseEvent e)  { label.setText("Click: " + ++n); }
      public void mouseEntered(MouseEvent e)  {}
      public void mouseExited(MouseEvent e)   {}
      public void mousePressed(MouseEvent e)  {}
      public void mouseReleased(MouseEvent e) {}

    });
```

Simple GUI

```
    JButton red = new JButton("Rosso");
    this.add(red, BorderLayout.NORTH);
    red.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e) {
        label.setForeground(Color.RED);
      }
    });

    JButton green = new JButton("Verde");
    this.add(green, BorderLayout.SOUTH);
    green.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e) {
        label.setForeground(Color.GREEN);
      }
    });

    JButton blue = new JButton("Blu");
    ... (BLUE) ...
    JButton black = new JButton("Nero");
    ... (BLACK) ...

    this.pack();
} }
```

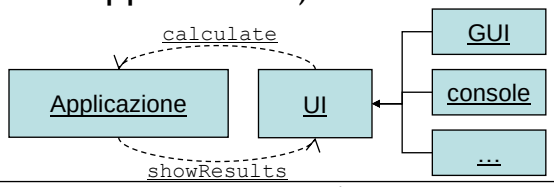

Matematica

- Vogliamo realizzare una semplice calcolatrice:

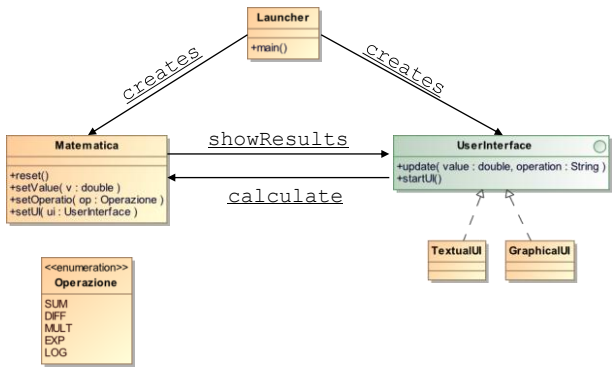


Matematica

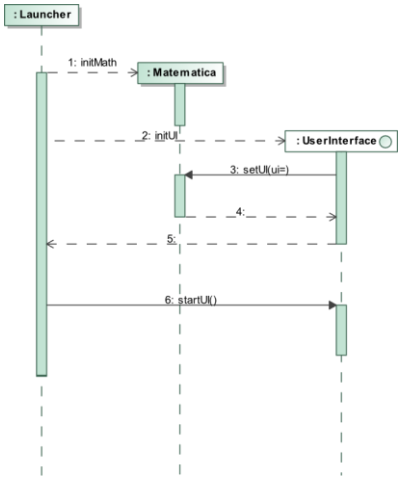
- L'applicazione è composta da interfaccia grafica e logica applicativa.
- Esiste un "Launcher" che inizializza sia la logica applicativa che l'interfaccia grafica e le collega
- È il primo passo verso la creazione di applicazioni multi-piattaforma (swing, interfacce web, interfacce testuali, tutte verso la stessa applicazione)



Matematica: progettazione



Matematica: inizializzazione



Matematica

```
public enum Operazione {
  Sum, Diff, Mult, Pow, Log;

  public double perform(double a, double b) {
    switch (this) {
      case Sum : return a + b;
      case Diff: return a - b;
      case Mult: return a * b;
      case Pow : return Math.pow(a, b);
      case Log : return Math.log(a) / Math.log(b);
    }

    return Double.NaN;
  }
}
```

- Esercizio:
  - Implementare le operazioni utilizzando il polimorfismo

Matematica

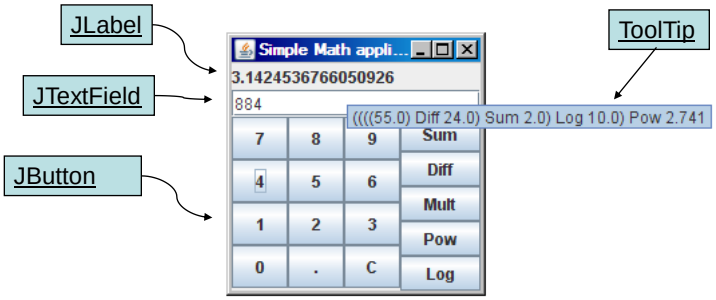
```
public class Matematica {
  private double result;  /* ultimo risultato */
  private Operazione op;  /* eventuale operazione selezionata */
  private String formula; /* formula calcolata */
  private UserInterface userInterface;

  public Matematica() { this.userInterface = null; reset(); }

  void setUserInterface(UserInterface userInterface) { this.userInterface = userInterface; }

  public void reset() {
    this.result = 0; this.op = null; this.formula = "";
    updateUI();
  }

  private void updateUI() { if (this.userInterface != null) this.userInterface.update(result, formula); }

  public void setOperation(Operazione op) { this.op = op; }

  public void setValue(double value) {
    if (op == null) { // Non c'è nessuna operazione attiva: imposta il valore
      this.formula = Double.toString(value);
      this.result = value;
    } else { // effettua l'operazione desiderata
      this.formula = "(" + this.formula + ") " + op.toString() + " " + value;
      this.result = op.perform(result, value);
      this.op = null;
    }

    updateUI();
  }
}
```


Matematica

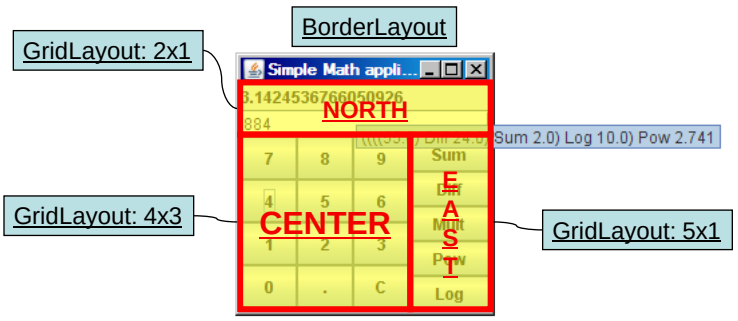
```
public class TextUserInterface implements UserInterface {

  private Matematica math;

  public TextUserInterface(Matematica math) {  this.math = math; math.setUserInterface(this);  }

  public void update(double result, String operation) { System.out.println(operation + " = " + result); }

  public void startUI() {
    Scanner s = new Scanner(System.in);

    System.out.println(" Dammi il valore:"); double value = s.nextDouble();
    math.setValue(value);

    while (true) {
      /* stampa il menu: 0 esci, 1 somma, 2 sottrai, 3 moltiplica, 4 exp, 5 log, 9 reset */
      int azione = s.nextInt();

      if (azione == 0) break;
      else if (azione == 1) math.setOperation(Operazione.Sum);
      else if (azione == 2) math.setOperation(Operazione.Diff);
      else if (azione == 3) math.setOperation(Operazione.Mult);
      else if (azione == 4) math.setOperation(Operazione.Pow);
      else if (azione == 5) math.setOperation(Operazione.Log);
      else if (azione == 9) math.reset();
      else {
        System.out.println("Azione non valida!");
        continue;
      }

      System.out.println(" Dammi il valore:"); value = s.nextDouble();
      mat
} } }
```

Live Demo

Matematica & Swing: Elementi



Matematica & Swing: Layout



Matematica: Graphical UI


```
public class GraphicalUserInterface
    extends JFrame implements UserInterface {

  private Matematica math;
  private JLabel risultato;
  private JTextField value;

  public GraphicalUserInterface(Matematica math) {
    super("Simple Math application");
    this.math = math;
    setup();

    math.setUserInterface(this);
  }

  public void startUI() {
    this.setVisible(true);
  }

  public void update(double result, String operation) {
    risultato.setText(Double.toString(result));
    risultato.setToolTipText(operation);
  }
```


Matematica: Graphical UI


```
private void setup() {
    this.setLayout(new BorderLayout());

    JPanel sopra = new JPanel(new GridLayout(2, 1));
    risultato = new JLabel("0"); sopra.add(risultato);
    value = new JTextField("0"); sopra.add(value);
    this.add(sopra, BorderLayout.NORTH);

    JPanel centro = new JPanel(new GridLayout(4, 3));
    // genera tutti i bottoni da 0 a 9
    JButton jbreset = new JButton("C");
    jbreset.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent arg0) {
        value.setText(""); math.reset();
      }
    });
    centro.add(jbreset);
    this.add(centro, BorderLayout.CENTER);

    JPanel operations = new JPanel(new GridLayout(5, 1));
    // genera i bottoni per le azioni: vedi prox slide
    this.add(operations, BorderLayout.EAST);

    this.pack();
  }

}
```

Matematica: Graphical UI

```
private void setup() {
    … (codice dalla slide precedente) …
    JPanel operations = new JPanel(new GridLayout(5, 1));

    JButton jbadd = new JButton("Add");
    jbadd.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        Matematica m = GraphicalUserInterface.this.math;
        m.setValue(Double.parseDouble(value.getText()));
        m.setOperation(Operazione.Sum);

       value.setText("");
      }
    });
    operations.add(jbadd);

    … (codice dalla slide precedente) …
}
```

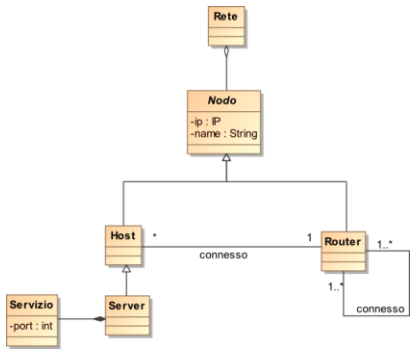

Network

Progettare un insieme di classi in grado di rappresentare una rete di computer.

Questa si compone di nodi, caratterizzati da un indirizzo IP e da un nome. I nodi possono essere di due tipi: host e router. Gli host sono connessi ad esattemente un router, mentre i router possono essere connessi ad un numero qualunque di host e ad almeno un altro router.

Esistono particolari tipi di host in grado di offrire servizi. Ogni servizio è erogato su una particolare porta di un determinato host.

Network


Rete
Nodo
-ip : IP
-name : String
Host
*
1
Router
1..*
connesso
1..*
connesso
Servizio
-port : int
Server