Appendici

Appendice A
Codice degli esempi

In questa appendice presentiamo il codice degli esempi principali utilizzati nei capitoli del libro. L'esempio introduttivo non viene ripreso perché già presente in modo completo. Per quanto riguarda invece l'esempio delle valute utilizzato per mostrare lo sviluppo con JUnit, verrà ripresa la prima versione presentata nel libro, con dispatching manuale, compresa la funzionalità di moltiplicazione introdotta per mezzo di TDD. La versione con utilizzo del pattern Visitor è completamente spiegata nel capitolo 24. Per chi fosse interessato alla variante originale con Visitor, da me modificata per ottenere maggiore trasparenza in IMoney, si rimanda invece all'implementazione distribuita con JUnit.

Applicare i pattern

La prima classe, Reader, rappresenta la classe principale. Ha lo scopo di centralizzare tutte le varianti scelte. Sarà l'unica classe a dover essere modificata, nel caso in cui si desiderassero altre configurazioni.

```
public class Reader
{
  public static void main (String args[]){
    String fileName= args[0];
    String outFile1= fileName + ".copy1";
    String outFile2= fileName + ".copy2";
```

```
        Broadcaster bc = new Broadcaster();
        LineWriter lw0 = new LineReverseWriter(System.out,new UpperCaseConvStrategy());
        LineWriter lw1 = new LineReverseWriter(outFile1,new NullConvStrategy());
        LineWriter lw2 = new LineStraightWriter(outFile2,new NullConvStrategy());
        bc.addPrintStream(lw0);
        bc.addPrintStream(lw1);
        bc.addPrintStream(new StatLineWriterDecorator(new StatLineWriterDecorator(lw2,'a'), 'b'));
        Copy cp = new Copy(new RealLineReader(fileName));
        cp.toOutput(bc);
    }
}


public class Copy
{
    protected List fStorage;

    public Copy (LineReader in){
        fStorage = new ArrayList();
        in.readAllLines(fStorage);
    }

    public void toOutput(LineWriter out){
        out.printAllLines(fStorage);
    }
}


public interface LineReader
{
    void readAllLines (List storage);
}

public class RealLineReader implements LineReader
{
    BufferedReader fReader;

    public RealLineReader(String identifier) {
        try{
            fReader = new BufferedReader(new FileReader(identifier));
        }catch (IOException e){
            e.printStackTrace();
        }
    }

    public void readAllLines (List storage) {
      try{
        String line = fReader.readLine();
        while (line != null){
          storage.add(line);
          line= fReader.readLine();
        }
      }catch (IOException e) {
        e.printStackTrace();
      }
    }
  }
```

Template e Factory Method

La parte variabile di Template, è rappresentata in questo caso unicamente dal metodo getIterator(), che viene implementato nelle classi finali LineStraightWriter e LineReverseWriter. Il metodo racchiude la creazione di un oggetto di conversione: si tratta quindi di un metodo factory. Con questa struttura otteniamo perciò la combinazione di due pattern: Template e Factory Method.

```
  public interface LineWriter
  {
    Iterator getIterator(List storage);
    void printAllLines(List storage);
    void printOneLine (String line);
  }

  public abstract class RealLineWriter implements LineWriter
  {
    private ConversionStrategy fConverter;
    private PrintWriter fOutput;

    public RealLineWriter (PrintStream stream) {
      fOutput = new PrintWriter(stream,true);
      fConverter = new NullConvStrategy();
    }

    public RealLineWriter (String identifier) {
      try {
        fOutput = new PrintWriter(new FileWriter(identifier),true);
      }catch (IOException e) {
         e.printStackTrace();
      }
      fConverter = new NullConvStrategy();
   }

   public void printAllLines(List storage) {
      Iterator iter=getIterator(storage);
      while (iter.hasNext()){
         printOneLine(iter.next().toString());
      }
   }

   public void printOneLine (String line) {
      fOutput.println(fConverter.convert(line));
   }

   ConversionStrategy getConverter() {
      return fConverter;
   }

   public void setConverter (ConversionStrategy cs) {
      fConverter = cs;
   }
}

public class LineStraightWriter extends RealLineWriter
{
   public LineStraightWriter(PrintStream out) {
      super(out);
   }

   public LineStraightWriter(PrintStream out, ConversionStrategy conversion) {
      super(out);
      setConverter(con);
   }

   public LineStraightWriter(String identifier, ConversionStrategy conversion) {
      super(identifier);
      setConverter(con);
   }

   public Iterator getIterator(List storage) {
        return storage.iterator();
    }
}

public class LineReverseWriter extends RealLineWriter
{
    public LineReverseWriter (PrintStream out) {
        super(out);
    }

    public LineReverseWriter (PrintStream out, ConversionStrategy conversion) {
        super(out);
        setConverter(conversion);
    }

    public LineReverseWriter (String identifier, ConversionStrategy conversion) {
        super(identifier);
        setConverter(conversion);
    }

    public Iterator getIterator(List storage) {
        return new ReverseIterator(storage);
    }
}
```

Decorator

```
public abstract class LineWriterDecorator implements LineWriter
{
    protected LineWriter fLWriter;

    LineWriterDecorator(LineWriter lw){
        fLWriter = lw;
    }

    public void printAllLines(List storage){
        fLWriter.printAllLines(storage);
    }

    public void printOneLine(String line){
        fLWriter.printOneLine(line);
    }

    public Iterator getIterator(List storage){
      return fLWriter.getIterator(storage);
   }
}

public class StatLineWriterDecorator extends LineWriterDecorator
{
   char fToFind;

   public StatLineWriterDecorator(LineWriter lw, char ch){
      super(lw);
      fToFind = ch;
   }

   protected int nrOfCharsInLine(String line){
      int found = 0;
      for (int i=0; i<line.length();i++){
         if (line.charAt(i)==fToFind)
            found++;
      }
      return found;
   }

   protected void makeStatistics(List storage){
      int countChar = 0;
      Iterator iter = getIterator(storage);
      String str;
      while(iter.hasNext()){
         str=(String)iter.next();
         countChar += nrOfCharsInLine(str);
      }
      printOneLine("\n");
      printOneLine("Numero totale di " + fToFind + ": " + countChar);
      printOneLine("---------------------------");
   }

   public void printAllLines(List storage) {
      super.printAllLines(storage);
      makeStatistics(storage);
   }
}
```

Observer

```
public class Broadcaster implements LineWriter
{
    protected List fObservers = new ArrayList();

    public void addPrintStream(LineWriter stream) {
        fObservers.add(stream);
    }

    public Iterator getIterator(List storage){
        return null;
    }

    public void printAllLines(List storage) {
        Iterator iter = fObservers.iterator();
        LineWriter lw;
        while (iter.hasNext()){
            lw = (LineWriter)iter.next();
            lw.printAllLines(storage);
        }
    }

    public void printOneLine(String line)
        Iterator iter = fObservers.iterator();
        LineWriter lw;
        while (iter.hasNext()){
            lw = (LineWriter)iter.next();
            lw.printOneLine(line);
        }
    }
}
```

Strategy

```
public interface ConversionStrategy
{
    public abstract String convert(String source);
}

public class NullConvStrategy implements ConversionStrategy
{
    public String convert(String source) {
        return source;
    }
}
```

```
public class UpperCaseConvStrategy implements ConversionStrategy
{
   public String convert(String source) {
      return source.toUpperCase();
   }
}
```

Pattern Half-Object

Si tratta dell'esempio mostrato a tappe durante la spiegazione del pattern Half-Object, anche chiamato HOPP. Si tratta del pattern che sta alla base dell'architettura di ULC.

Lato server

```
public class FTPServer
{
   private File fCurrentDir = new File(".");
   private String fLastOperation = null;

   public String chdir(String dir) {
      File newDir = new File(fCurrentDir, dir);
      if (newDir.isDirectory()) {
         fCurrentDir = newDir;
         try {
            return fCurrentDir.getCanonicalPath();
         } catch (IOException e) {
            return null;
         }
      }
      return null;
   }

   public String[] dir() {
      return fCurrentDir.list();
   }

   public String getLastOperation(){
      return fLastOperation;
   }

   protected void setLastOperation(String lastOperation) {
    fLastOperation = lastOperation;
  }
}


public class ProtocolProxy extends Thread
{
  private Socket fIncoming;
  private FTPServer fFTPServer = new FTPServer();

  public ProtocolProxy(Socket incoming) {
    fIncoming = incoming;
  }

  public void run() {
    BufferedReader reader = null;
    try {
      reader = new BufferedReader(new InputStreamReader(fIncoming.getInputStream()));
      PrintWriter writer = new PrintWriter(fIncoming.getOutputStream(), true);
      while (true) {
        String line = reader.readLine();
        if (line == null){
          break;
        }
        if (line.startsWith("cd")) {
          doChangeDir(cutOperation(line), writer);
        } else if (line.startsWith("dir")) {
          doDir(writer);
        } else if (line.startsWith("last")) {
          doLastOperation(cutOperation(line));
        } else if (line.startsWith("exit")) {
          break;
        } else {
          writer.println("Invalid operation");
        }
      }
    } catch (Exception e) {
      e.printStackTrace();
    } finally {
      try {
        reader.close();
      } catch (Exception ignore) {
      }
    }
  }

  private String cutOperation(String operation) throws Exception {
    int start = operation.indexOf(" ");
    if (start == -1){
      throw new Exception("Wrong formatted operation");
    }
    return operation.substring(start).trim();
  }

  private void doChangeDir(String argument, PrintWriter writer) {
    String newDir = fFTPServer.chdir(argument);
    if (newDir != null) {
      writer.println("OK: " + newDir);
    } else {
      writer.println("Cannot cange into directory: " + argument);
    }
  }

  private void doDir(PrintWriter writer) {
    String[] fileNames = fFTPServer.dir();
    if (fileNames != null) {
      writer.println("OK");
      writer.println(fileNames.length);
      for (int i = 0; i < fileNames.length; i++) {
        writer.println(fileNames[i]);
      }
    } else {
      writer.println("No Files in this directory");
    }
  }

  private void doLastOperation(String argument) {
    fFTPServer.setLastOperation(argument);
  }
}

public class ServerStart
{
  public static void main(String[] args) throws IOException {
    ServerSocket server = new ServerSocket(7007);
    while (true){
            try {
                Socket incoming = server.accept();
                new ProtocolProxy(incoming).start();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Lato Client

```
public class FTPClient
{
    private Socket fSocket;
    private BufferedReader fReader;
    private PrintWriter fWriter;
    private String fLastOperation;

    public FTPClient(String server, int port) throws IOException{
        fSocket = new Socket(server,port);
        fReader = new BufferedReader(new InputStreamReader(fSocket.getInputStream()));
        fWriter = new PrintWriter(fSocket.getOutputStream(),true);
    }

    public String chdir(String dir) throws IOException{
        fWriter.println("cd " + dir);
        String response = fReader.readLine();
        if (response.startsWith("OK")){
            setLastOperation("chdir");
             return response.substring(response.indexOf(" "));
        }else{
            return null;
        }
    }

    public String[] dir() throws IOException {
        fWriter.println("dir");
        fWriter.flush();
        String response = fReader.readLine();
        if (response.startsWith("OK")){
            setLastOperation("dir");
            int quantity = Integer.parseInt(response = fReader.readLine());
            List files = new ArrayList(quantity);
            for (int i=0; i<quantity; i++){
               files.add(response = fReader.readLine());
            }
            return (String[]) files.toArray(new String[files.size()]);
         }
         return null;
      }

      public String getLastOperation(){
         return fLastOperation;
      }

      public void exit() throws IOException {
         fWriter.println("exit");
         fReader.close();
         fWriter.close();
         fSocket.close();
      }

      private void setLastOperation(String lastOperation) {
         fLastOperation = lastOperation;
         fWriter.println("last " + fLastOperation);
      }
   }


   public class GUITest
   {
      private FTPClient fFTPClient;

      public GUITest(String server, int port) {
         try {
            fFTPClient = new FTPClient(server, port);
         } catch (IOException e) {
            System.err.println("No server connection could be established");
         }
      }

      public void loop() {
         BufferedReader console = new BufferedReader(new InputStreamReader(System.in));
         while (true) {
```

```
      String line = null;
      try {
        System.out.println("Enter request: ");
        line = console.readLine();
        System.out.println("Request: " + line);
      } catch (IOException e) {
        try {
          fFTPClient.exit();
        } catch (Exception ignore) {}
        e.printStackTrace();
        System.exit(1);
      }
      try {
        if (line.startsWith("dir")) {
          callClientDir();
        } else if (line.startsWith("cd")) {
          callClientChangeDir(cutOperation(line));
        } else if (line.equals("exit")) {
          try {
            fFTPClient.exit();
          } catch (Exception ignore) {}
          System.exit(0);
        } else{
          System.out.println("Unrecognized command on FTPClient");
        }
        System.out.println("Last Operation: " + fFTPClient.getLastOperation());
      }catch (Exception e) {
        System.out.println("The operation could not be completed");
        e.printStackTrace();
      }
    }
  }

  private void callClientChangeDir(String dir) {
    try {
      String newDir = fFTPClient.chdir(dir);
      if (newDir != null){
        System.out.println("chdir OK: " + newDir);
      }else{
        System.out.println("chdir failed");
      }
    } catch (IOException e) {
      System.out.println("The operation could not be completed");
      }
    }

    private void callClientDir() {
      String[] fileNames = new java.lang.String[0];
      try {
        fileNames = fFTPClient.dir();
        if (fileNames != null) {
          for (int i = 0; i < fileNames.length; i++) {
            System.out.println(fileNames[i]);
          }
        }
      } catch (IOException e) {
        System.out.println("The operation could not be completed");
      }
    }

    private String cutOperation(String operation) throws Exception {
      int start = operation.indexOf(" ");
      if (start == -1){
        throw new Exception("Wrong formatted operation");
      }
      return operation.substring(start);
    }

    public static void main(String[] args) {
      GUITest ui = new GUITest("localhost",7007);
      ui.loop();
    }
  }
```

Gestione di valute

Viene mostrato il codice della versione con dispatching manuale. Per quanto riguarda i test, si rimanda al capitolo 18.

Per la versione con utilizzo di Visitor, rimando al capitolo 24, oppure alla distribuzione di JUnit, con però particolare attenzione alla gerarchia di classi, che è diversa nelle due varianti. La classe AbstractMoney usata nella versione Visitor del libro è infatti fondamentale per rendere maggiormente trasparente l'utilizzo di IMoney.add().

```
public interface IMoney
{
   public IMoney add(IMoney aMoney);
   public IMoney multiply(int factor);
}

public abstract class AbstractMoney implements IMoney
{
   public IMoney add(IMoney m){
      if (m instanceof Money){
         return addMoney((Money)m);
      }else{
         return addMoneyBag((MoneyBag)m);
      }
   }

   abstract IMoney addMoney(Money m);
   abstract IMoney addMoneyBag(MoneyBag b);
}

class Money  extends AbstractMoney
{
   private int fAmount;
   private String fCurrency;

   public Money(int amount, String currency) {
      fAmount= amount;
      fCurrency= currency;
   }

   public int getAmount() {
      return fAmount;
   }

   public String getCurrency() {
      return fCurrency;
   }

   public IMoney addMoney(Money m) {
      if (m.getCurrency().equals(getCurrency())){
         return new Money(getAmount()+m.getAmount(), getCurrency());
      }
      return new MoneyBag(this, m);
   }

   public IMoney addMoneyBag(MoneyBag b) {
      return b.addMoney(this);
   }

   public boolean equals(Object other) {
      return
         ((other instanceof Money) &&
         ((Money)other).getCurrency().equals(getCurrency()) &&
         ((Money)other).getAmount() == getAmount());
   }

   public IMoney multiply(int factor){
      return new Money(getAmount() * factor, getCurrency());
   }
}


class MoneyBag extends AbstractMoney {
   private List fMonies = new ArrayList();


   public MoneyBag(Money m1, Money m2) {
      appendMoney(m1);
      appendMoney(m2);
   }


   public MoneyBag(Money bag[]) {
      for (int i = 0; i < bag.length; i++){
         appendMoney(bag[i]);
      }
   }


   public MoneyBag(Money m, MoneyBag mb) {
      appendMoneyBag(mb);
      appendMoney(m);
   }


   public MoneyBag(MoneyBag mb1, MoneyBag mb2) {
      appendMoneyBag(mb1);
      appendMoneyBag(mb2);
   }


   public void appendMoney(Money m) {
      if (m.getAmount() != 0) {
         int position = findMoneyWithSameCurrency(fMonies, m.getCurrency());
         if (position == -1){
            fMonies.add(m);
         }else {
            Money existingMoney = (Money) fMonies.remove(position);
            Money result = (Money) m.addMoney(existingMoney);
            appendMoney(result);
         }
      }
   }

   public boolean equals(Object mb) {
      if ((mb instanceof MoneyBag) && (fMonies.size() == ((MoneyBag) mb).getMoneyList().size())){
         List moneyListFromMoneyBag = ((MoneyBag) mb).getMoneyList();
         for (int i = 0; i < fMonies.size(); i++) {
            if (moneyListFromMoneyBag.indexOf(fMonies.get(i)) == -1) {
               return false;
            }
         }
         return true;
      }
      return false;
   }

   public IMoney addMoney(Money m) {
      return new MoneyBag(m, this).simplify();
   }

   public IMoney addMoneyBag(MoneyBag b) {
      return new MoneyBag(b, this).simplify();
   }

   public IMoney multiply(int factor){
      List tempResults = new ArrayList();
      Iterator iter = getMoneyList().iterator();
      while(iter.hasNext()){
         Money money = (Money)iter.next();
         tempResults.add(money.multiply(factor));
      }
      return new MoneyBag((Money[])tempResults.toArray(new Money[tempResults.size()]));
   }

   private List getMoneyList() {
      return fMonies;
   }

   private void appendMoneyBag(MoneyBag mb) {
        Iterator iter = mb.getMoneyList().iterator();
        while (iter.hasNext()) {
            appendMoney((Money) iter.next());
        }
    }

    private int findMoneyWithSameCurrency(List monies, String currency) {
        for (int i = 0; i < monies.size(); i++) {
            Money money = (Money) monies.get(i);
            if (money.getCurrency().equals(currency)){
                return i;
            }
        }
        return -1;
    }

    private IMoney simplify() {
        if (fMonies.size() == 1){
            return (IMoney) fMonies.get(0);
        }
        return this;
    }
}
```

Separate Domain from Presentation

Programma principale

Il programma principale consiste in un pannello di controllo con due pulsanti: uno per aprire la prima view, un altro per aprire una view del secondo tipo. Si tratta quindi di un programma principale di dimostrazione, che permette facilmente di verificare come tutte le view si rifacciano allo stesso, unico modello.

```
public class Main extends JFrame
{
    private JButton fTextWin,fSliderWin;
    private Model fSharedModel;

    public Main(String title){
        super(title);
        fSharedModel = new Model();
        fTextWin = new JButton("Text");
        fTextWin.addActionListener(new NewTextWinAction());
        fSliderWin = new JButton("Slider");
        fSliderWin.addActionListener(new NewSliderWinAction());
        getContentPane().setLayout(new GridLayout(1,2));
        getContentPane().add(fTextWin);
        getContentPane().add(fSliderWin);
    }

    private class NewTextWinAction implements ActionListener
    {
        public void actionPerformed(ActionEvent event){
            JFrame frame = new View1("View 1: TextFields",fSharedModel);
            frame.pack();
            frame.setVisible(true);
        }
    }

    private class NewSliderWinAction implements ActionListener
    {
        public void actionPerformed(ActionEvent event){
            JFrame frame = new View2("View 2: Slider",fSharedModel);
            frame.pack();
            frame.setVisible(true);
        }
    }

    public static void main(String[] args){
        JFrame frame = new Main("Main");
        frame.addWindowListener(new WindowAdapter () {
            public void windowClosing(WindowEvent e){
                System.exit(0);
            }
        });
        frame.pack();
        frame.setVisible(true);
    }
}
```

Modello

```
public class Model
{
    private List fObservers = new ArrayList();
    private String fStart, fEnd, fDifference;
```

```
public Model(){
   fStart = "0";
   fEnd = "0";
   fDifference = "0";
}

public void addObserver(Observer observer){
   fObservers.add(observer);
}

protected void  notifyObservers(){
   Iterator iter = fObservers.iterator();
   while(iter.hasNext()){
      Observer ob = (Observer)iter.next();
      ob.update(this);
   }
}

public String getStart(){
   return fStart;
}

public void setStart(String arg){
   fStart = arg;
   notifyObservers();
}

public String getEnd(){
   return fEnd;
}

public void setEnd(String arg){
   fEnd = arg;
   notifyObservers();
}

public String getDifference(){
   return fDifference;
}

public void setDifference(String arg){
   fDifference = arg;
   notifyObservers();
```

```
    }

    public void computeDifference(){
        int start = Integer.parseInt(getStart());
        int end = Integer.parseInt(getEnd());
        setDifference(String.valueOf(end - start));
    }

    public void computeEnd(){
        int start = Integer.parseInt(getStart());
        int length= Integer.parseInt(getDifference());
        setEnd(String.valueOf(start + length));
    }
}
```

View

Rispetto alla versione presentata nel capitolo 23, qui i due diversi tipi di visualizzazione hanno parte della funzionalità in una classe astratta comune (AbstractView). Abbiamo quindi applicato il refactoring Extract Superclass.

```
public interface Observer {
    public void update(Model model);
}

public abstract class AbstractView extends JFrame implements Observer
{
    private Model fModel;

    public AbstractView (String title, Model mod){
        super(title);
        fModel = mod;
        fModel.addObserver(this);
    }

    public void computeEnd(){
        fModel.computeEnd();
    }

    public void computeDifference(){
        fModel.computeDifference();
    }
```

```
  public String getStart(){
    return fModel.getStart();
  }

  public void setStart(String arg){
    fModel.setStart(arg);
  }

  public String getEnd(){
    return fModel.getEnd();
  }

  public void setEnd(String arg){
    fModel.setEnd(arg);
  }

  public String getDifference(){
    return fModel.getDifference();
  }

  public void setDifference(String arg){
    fModel.setDifference(arg);
  }

  protected boolean isNotInteger(String text){
    return false;
  }
}


public class View1 extends AbstractView
{
  private JTextField fStartField, fEndField, fDifferenceField;

  public View1(String title, Model model){
    super(title,model);
    getContentPane().setLayout(new GridLayout(3,1));
    fStartField = new JTextField("0");
    fEndField = new JTextField("0");
    fDifferenceField = new JTextField("0");
    fStartField.setPreferredSize(new Dimension(200,20));
    fStartField.addFocusListener(new StartFocus());
    fStartField.addActionListener(new StartAction());
    fEndField.addFocusListener(new EndFocus());
    fEndField.addActionListener(new EndAction());
    fDifferenceField.addFocusListener(new DifferenceFocus());
    fDifferenceField.addActionListener(new DifferenceAction());
    getContentPane().add(fStartField);
    getContentPane().add(fEndField);
    getContentPane().add(fDifferenceField);
    update(model);
  }

  public void update(Model model){
    fStartField.setText(getStart());
    fEndField.setText(getEnd());
    fDifferenceField.setText(getDifference());
  }

  private void startCheck(){
    setStart(fStartField.getText());   //necessrio per aggiornare il model
    if (isNotInteger(getStart())){
      setStart("0");
    }
    computeDifference();
  }

  private void endCheck(){
    setEnd(fEndField.getText());
    if (isNotInteger(getEnd())){
      setEnd("0");
    }
    computeDifference();
  }

  private void differenceCheck(){
    setDifference(fDifferenceField.getText());
    if (isNotInteger(getDifference())){
      setDifference("0");
    }
    computeEnd();
  }

  private class StartFocus extends FocusAdapter
  {
    public void focusLost(FocusEvent event){
            startCheck();
        }
    }

    private class StartAction implements ActionListener
    {
        public void actionPerformed(ActionEvent event){
            startCheck();
        }
    }

    private class EndFocus extends FocusAdapter
    {
        public void focusLost(FocusEvent event){
            endCheck();
        }
    }

    private class EndAction implements ActionListener
    {
        public void actionPerformed(ActionEvent event){
            endCheck();
        }
    }

    private class DifferenceFocus extends FocusAdapter
    {
        public void focusLost(FocusEvent event){
            differenceCheck();
        }
    }

    private class DifferenceAction implements ActionListener
    {
        public void actionPerformed(ActionEvent event){
            differenceCheck();
        }
    }
}


public class View2 extends AbstractView
{
```

```
private final static int MIN = -40, MAX = 40;
private JSlider fStartSlider, fEndSlider, fDifferenceSlider;

public View2(String title, Model model){
   super(title,model);
   getContentPane().setLayout(new GridLayout(3,1));
   fStartSlider = createSlider();
   fEndSlider = createSlider();
   fDifferenceSlider = createSlider();
   fStartSlider.addChangeListener(new StartChange());
   fEndSlider.addChangeListener(new EndChange());
   fDifferenceSlider.addChangeListener(new DifferenceChange());
   getContentPane().add(fStartSlider);
   getContentPane().add(fEndSlider);
   getContentPane().add(fDifferenceSlider);
   update(model);
}

private JSlider createSlider(){
   JSlider temp = new JSlider(JSlider.HORIZONTAL, MIN, MAX, 0);
   temp.setMajorTickSpacing(10);
   temp.setMinorTickSpacing(1);
   temp.setPaintTicks(true);
   temp.setPaintLabels(true);
   return temp;
}

public void update(Model model){
   fStartSlider.setValue(Integer.parseInt(getStart()));
   fEndSlider.setValue(Integer.parseInt(getEnd()));
   fDifferenceSlider.setValue(Integer.parseInt(getDifference()));
}


private void startCheck(){
   setStart(String.valueOf(fStartSlider.getValue()));   //necessrio per aggiornare il model
   if (isNotInteger(getStart())){
      setStart("0");
   }
   computeDifference();
}

private void endCheck(){
        setEnd(String.valueOf(fEndSlider.getValue()));
        if (isNotInteger(getEnd())){
            setEnd("0");
        }
        computeDifference();
    }

    private void differenceCheck(){
        setDifference(String.valueOf(fDifferenceSlider.getValue()));
        if (isNotInteger(getDifference())){
            setDifference("0");
        }
        computeEnd();
    }

    private class StartChange implements ChangeListener
    {
        public void stateChanged(ChangeEvent e) {
            startCheck();
        }
    }

    private class EndChange implements ChangeListener
    {
        public void stateChanged(ChangeEvent e) {
            endCheck();
        }
    }

    private class DifferenceChange implements ChangeListener
    {
        public void stateChanged(ChangeEvent e) {
            differenceCheck();
        }
    }
}
```

Appendice B
Riferimenti bibliografici

[Beck-1997]
BECK K., *Smalltalk Best Practice Patterns*, Prentice Hall, 1997.

[Beck-1999]
BECK K., *Extreme Programming Explained*, Addison Wesley, 1999.

[Beck-2003]
BECK K., *Test-Driven Development*, Addison Wesley, 2003

[Bloch-2001]
BLOCH J., *Effective Java, Programming Language Guide*, The Java Series, 2001.

[Bulka-2000]
BULKA D., *Java, Performance and Scalability, Volume 1*, Addison Wesley, 2000

[Cotter-1995]
COTTER S. POTEL M., *Inside Taligent Technology*, Addison Wesley, 1995

[Flanagan-1999]
FLANAGAN D., *Java in a Nutshell*, 3rd Edition, 1999.

[Fowler-1999]
FOWLER M., *Improving the Design of Existing Code*, Addison Wesley, 1999

[Fowler-2002]
FOWLER M., *Patterns of Enterprise Application Architecture*, Addison Wesley, 2002

[Gabriel-1996]
GABRIEL P.R., *Patterns of Software: Tales From The Software Community*, Oxford University Press, 1996

[Gamma-1995]
GAMMA E., HELM R., JOHNSON R., VLISSIDES J., *Design Pattern: Elements of Reusable Object-Oriented Software*, Addison Wesley, 1995

[Gamma-2003]
GAMMA E., BECK K., *Contributing to Eclipse*, 2003

[Giulietti-1999]
GIULIETTI R., PEDRAZZINI S., *Thin Client for Web Using Swing*, Published in "Clemence Cap (Eds.): JIT-99, Lecture Notes in Computer Science", Springer Verlag, 1999.

[Govoni-1999]
GOVONI DERREN, *Java Application Frameworks*, John Wiley & Sons, 1999.

[Horstmann-1999]
HORSTMANN C.S., CORNELL G., *Java 2, I fondamenti*, Mc Graw Hill, 1999.

[Java-2001]
AA.VV., *Manuale Pratico di Java*, Hops, 2001.

[Kiczales-1991]
KICZALES G., DES RIVIÈRES J., BOBROW D.G., *The Art of the Metaobject Protocol,* The MIT Press, 1991.

[Meszaros-1995]
MESZAROS G., *Pattern: Half-Object+Protocol Pattern*, in Coplien, Schmidt: "Patterns Languages of Program Design", Addison-Wesley, Reading 1995.

[Pedrazzini-2001]
PEDRAZZINI S., *A caccia di pattern*, Mokabyte, marzo/aprile, 2001.

[Pedrazzini-2004]
PEDRAZZINI S., *Java: la piattaforma ideale per architetture "rich thin client"*, Mokabyte, settembre 2004.

[Reeves-1992]
REEVES J.W., *What is Software Design?*, C++ Journal, 1992

[Vetti-Tagliati-2001]
VETTI TAGLIATI L., *UML e ingegneria del software*, Tecniche Nuove, 2001.

[Wirth-1987]
WIRTH N., *From Modula to Oberon,* "ETH Report 82", Institut für Informatik, ETH Zürich, 1987.

[agile-manifesto]
http://www.agilemanifesto.org

[Canoo.net]
http://www.canoo.net

[clos-unit]
http://www.dti.supsi.ch/~pedrazz/clos-unit

[Fowler]
http://www.martinfowler.com

[Jacaranda]
http://www.dti.supsi.ch/~pedrazz/jacaranda

[JUnit]
http://www.junit.org

Indice analitico