Patró Observador

Jordi Molist Pladevall

Presentació del patró observador

- És útil per dissenyar un model de comunicació entre un conjunt d'objectes que depenen d'un altre.
  - Els dependents seran els observers
  - I l'objecte observat serà el subject.

Presentació del patrò observador

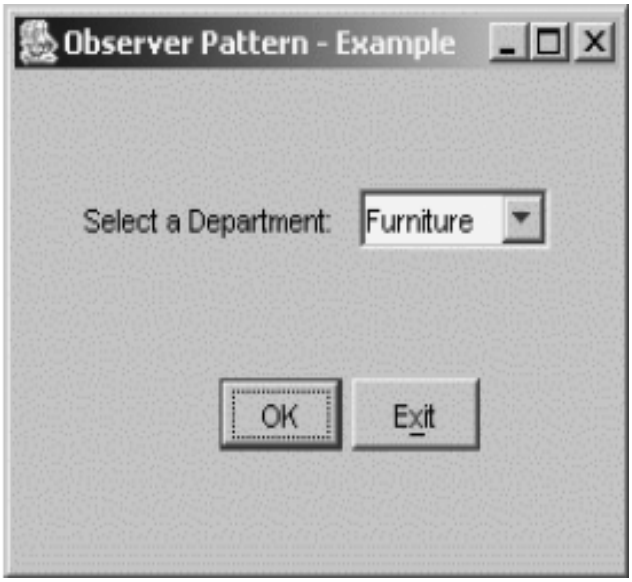
- Un observador necessita saber l'estat del seu *subject.*
- Un observador:
  - Registrar quan necessita informació d'aquest.
  - Ha de desregistrar-se per desvincular-se del *subject.*
- Un *subject* ha de proporcionar una *interface* que ho permeti.

Presentació del patrò observador

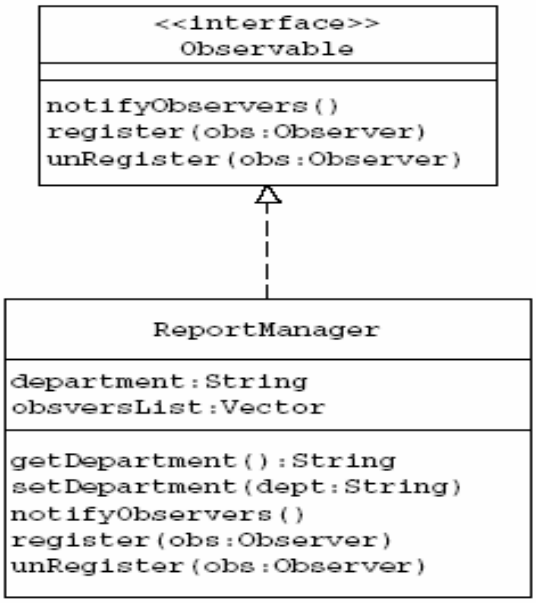
- El patró ha de seguir un dels dos models:
  - Pull -> L'observador va preguntant al subject la informació del seu estat.
  - Push -> El subject envia als observadors el seu estat.

Exemple

- L'exemple tracta sobre una aplicació que ofereix al director d'una botiga, saber les ventes que té cada departament d'aquesta.
- *Subject*, aplicació que s'ofereix al client per poder seleccionar com va cada departament.
- Observadors:
  - Monthly report
  - YTD sales chart



Codi del Subject




```
public class ReportManager extends JFrame
  implements Observable {
          …
          …
  private Vector observersList;
  private String department;
  public ReportManager() throws Exception {
          …
          …
    observersList = new Vector();
          …
          …
  }
  public void register(Observer obs) {
    //Add to the list of Observers
    observersList.addElement(obs);
  }
  public void unRegister(Observer obs) {
    //remove from the list of Observers
  }
```

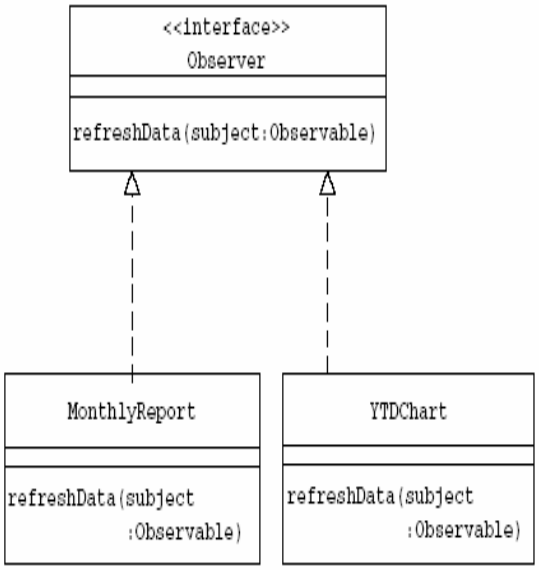
Codi del Subject

```
public void notifyObservers() {
  //Send notify to all Observers
  for (int i = 0; i < observersList.size(); i++) {
    Observer observer =
      (Observer) observersList.elementAt(i);
    observer.refreshData(this);
  }
}
public String getDepartment() {
  return department;
}
public void setDepartment(String dept) {
  department = dept;
}
```

Codi del subject

```
class ButtonHandler implements ActionListener {
  ReportManager subject;
  public void actionPerformed(ActionEvent e) {
    if (e.getActionCommand().equals(ReportManager.EXIT)) {
      System.exit(1);
    }
    if (e.getActionCommand().equals(ReportManager.SET_OK)) {
      String dept = (String)
                    cmbDepartmentList.getSelectedItem();
      //change in state
      subject.setDepartment(dept);
      subject.notifyObservers();
    }
  }
  public ButtonHandler() {
  }
  public ButtonHandler(ReportManager manager) {
    subject = manager;
  }
}
```

Codi observadors




```
public class MonthlyReport extends JFrame implements Observer {
        "
        "
 private ReportManager objReportManager;
 public MonthlyReport(ReportManager inp_objReportManager)
 throws Exception {
   super("Observer Pattern - Example");
   objReportManager = inp_objReportManager;
   objReportManager.register(this);
```

Codi Observador

```
public void refreshData(Observable subject) {
  if (subject == objReportManager) {
    //get subject's state
    String department = objReportManager.getDepartment();
    lblTransactions.setText(
      "Current Month Transactions - " +
      department);
    Vector trnList =
      getCurrentMonthTransactions(department);
    String content = "";
    for (int i = 0; i < trnList.size(); i++) {
      content = content +
                trnList.elementAt(i).toString() + "\n";
    }
    taTransactions.setText(content);
  }
}
```

Programa principal

```
public class SupervisorView {
        ..
        ..
  public static void main(String[] args) throws Exception {
    //Create the Subject
    ReportManager objSubject = new ReportManager();
    //Create Observers
    new MonthlyReport(objSubject);
    new YTDChart(objSubject);
  }
```

Diagrama de classes final

