Teaser: Padrões de Projeto em DSOA (AspectJ)

Eduardo Figueiredo
http://www.dcc.ufmg.br/~figueiredo
reuso.software@gmail.com

04 Abril 2012

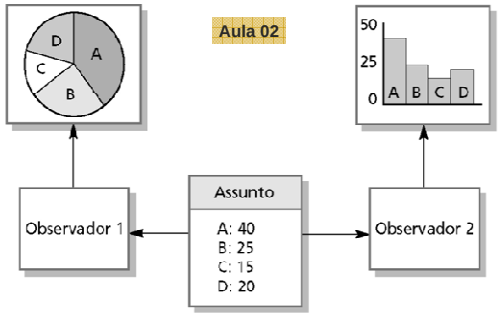
Padrões GoF

- Os 23 padrões foram originalmente propostos para desenvolvimento OO
  - Podem ser naturalmente implementados em linguagens OO, como Java
- Entretanto, os problemas descritos pelos padrões podem ocorrem no desenvolvimento não-OO
  - Qual seria a solução não-OO?

Padrões em AspectJ

- Hannemann e Kiczales desenvolveram soluções orientadas a aspectos para os 23 padrões GoF
  - Exemplos foram implementados em AspectJ
- Na avaliação de Hannemann e Kiczales
  - 17 do 23 padrões tem melhor modularidade em aspectos do que em objetos
  - 12 do 23 padrões tem melhor reuso de código em aspectos

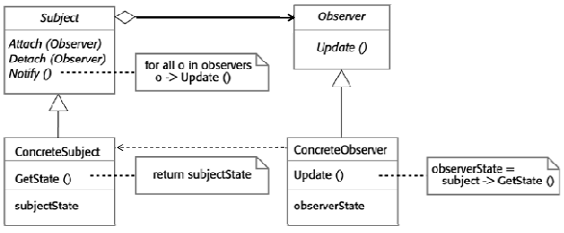
Exemplo: Padrão Observer



Padrão Obsever

- Nome
  - Observer
- Descrição do problema
  - Separa o objeto de sua forma de apresentação
- Descrição da solução (próximo slide)
- Consequências
  - Otimizações para melhorar a atualização da apresentação

Solução do Observer (OO)



Implementação Java (OO)

Interface que define o papel Subject

```
public interface ChangeSubject {
  public void addObserver(ChangeObserver o);

  public void removeObserver(ChangeObserver o);

  public void notifyObservers();
}
```

Interface que define o papel Observer

```
public interface ChangeObserver {
  public void refresh(ChangeSubject s);
}
```

Classe Screen (Solução OO)

```
public class Screen implements ChangeSubject, ChangeObserver {
  private HashSet observers = new HashSet();
  private String name;

  public Screen(String s) {
    this.name = s;
  }

  public void display (String s) {
    System.out.println(name + ": " + s);
    notifyObservers();
  }

  public void refresh(ChangeSubject s) {
    String subjectTypeName = s.getClass().getName();
    subjectTypeName = subjectTypeName.substring(
      subjectTypeName.lastIndexOf(".")+1, subjectTypeName.length());
    display("update received from a "+subjectTypeName+" object");
  }
}
```

```
public void addObserver(ChangeObserver o) {
  this.observers.add(o);
}

public void removeObserver(ChangeObserver o) {
  this.observers.remove(o);
}

public void notifyObservers() {
  for (Iterator e = observers.iterator() ; e.hasNext() ;) {
    ((ChangeObserver)e.next()).refresh(this);
  }
}
```

Classe Point (Solução OO)

```
public class Point implements ChangeSubject {
  private HashSet observers = new HashSet();
  private int x; private int y;
  private Color color;

  public Point(int x, int y, Color color) { ... }

  public int getX() {return x; }
  public int getY() { return y; }
  public Color getColor() { return color; }

  public void setX(int x) {
    this.x = x;
    notifyObservers();
  }

  public void setY(int y) {
    this.y = y;
    notifyObservers();
  }

  public void setColor(Color color) {
    this.color = color;
    notifyObservers();
  }
}
```

```
public void addObserver(ChangeObserver o) {
  this.observers.add(o);
}

public void removeObserver(ChangeObserver o) {
  this.observers.remove(o);
}

public void notifyObservers() {
  for (Iterator e = observers.iterator() ; e.hasNext() ;) {
    ((ChangeObserver)e.next()).refresh(this);
  }
}
```

Classe Screen (Solução OA)

```
public class Screen {
  private String name;

  public Screen(String s) {
    this.name = s;
  }

  public void display (String s) {
    System.out.println(name + ": " + s);
  }
}
```

Classe Point (Solução OA)

```
public class Point {
  private int x;
  private int y;
  private Color color;

  public Point(int x, int y, Color color) {
    this.x=x;
    this.y=y;
    this.color=color;
  }

  public int getX() { return x; }

  public int getY() { return y; }

  public void setX(int x) { this.x = x; }

  public void setY(int y) { this.y = y; }

  public Color getColor() { return color; }

  public void setColor(Color color) { this.color=color; }
}
```

Aspecto ObserverProtocol (OA)

```
public abstract aspect ObserverProtocol {
  protected interface Subject  { }
  protected interface Observer { }

  private WeakHashMap perSubjectObservers;

  protected List getObservers(Subject subject) {
    if (perSubjectObservers == null) {
      perSubjectObservers = new WeakHashMap();
    }
    List obs= (List)perSubjectObservers.get(subject);
    if ( obs == null ) {
      obs = new LinkedList();
      perSubjectObservers.put(subject, obs);
    }
    return obs;
  }

  public void addObserver(Subject subject, Observer observer) {
    getObservers(subject).add(observer);
  }

  public void removeObserver(Subject subject, Observer observer) {
    getObservers(subject).remove(observer);
  }
}
```

```
protected abstract pointcut subjectChange(Subject s);

after(Subject subject): subjectChange(subject) {
  Iterator iter = getObservers(subject).iterator();
  while ( iter.hasNext() ) {
    updateObserver(subject, ((Observer)iter.next()));
  }
}

protected abstract void updateObserver(
    Subject subject, Observer observer);
```

CoordinateObserver (OA)

```
public aspect CoordinateObserver extends ObserverProtocol{
  declare parents: Point  implements Subject;
  declare parents: Screen implements Observer;

  protected pointcut subjectChange(Subject subject):
    (call(void Point.setX(int)) ||
    call(void Point.setY(int)) ) && target(subject);

  protected void updateObserver(Subject subject, Observer observer) {
    ((Screen)observer).display("Screen updated "+
    "(point subject changed coordinates).");
  }
}
```

ColorObserver (OA)

```
public aspect ColorObserver extends ObserverProtocol{
  declare parents: Point  implements Subject;
  declare parents: Screen implements Observer;

  protected pointcut subjectChange(Subject subject):
    call(void Point.setColor(Color)) && target(subject);

  protected void updateObserver(Subject subject, Observer observer) {
    ((Screen)observer).display("Screen updated "+
    "(point subject changed color).");
  }
}
```

ScreenObserver (OA)

```
public aspect ScreenObserver extends ObserverProtocol{
  declare parents: Screen implements Subject;
  declare parents: Screen implements Observer;

  protected pointcut subjectChange(Subject subject):
    call(void Screen.display(String)) && target(subject);

  protected void updateObserver(Subject subject, Observer observer) {
    ((Screen)observer).display("Screen updated " +
    "(screen subject displayed message).");
  }
}
```

To be continue...

- Veremos mais detalhes em outras aulas
- Ler o artigo da bibliografia

Bibliografia da Aula

- Jan Hannemann and Gregor Kiczales. **Design Pattern Implementation in Java and AspectJ**. Proceedings of the 17th Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA), 2002.
  - Disponível no website da disciplina

Próxima Aula!

- Laboratório 2011?
  - Possivelmente
- Implementação em AspectJ