Тема: Наблюдатель (Observer)

Подготовили:
Махиня Д.А.
Кравченко В.С.

Что представляет собой паттерн Observer?

- Наблюдатель - паттерн поведения объектов, устанавливающий систему оповещения объектами своих соседей в процессе их деятельности.
- Паттерн-наблюдатель определяет зависимость типа «один ко многим» между объектами таким образом, что при изменении состояния одного объекта все зависящие от него оповещаются об этом и автоматически обновляются.

Главная идея

Основная идея заключается в том, что если какое-то действие происходит в одном классе, то оповещаются все классы, заинтересованные в данном изменении.

Ключевыми объектами являются *субъект и наблюдатель.*

У субъекта может быть сколько угодно зависимых от него наблюдателей.

Все наблюдатели уведомляются об изменениях в состоянии субъекта.

Получив уведомление, наблюдатель опрашивает субъект, чтобы синхронизировать с ним свое состояние.

Такого рода взаимодействие часто называется отношением издатель. Субъект издает или публикует уведомления и рассылает их, даже не имея информации о том, какие объекты являются подписчиками. На получение уведомлений может подписаться неограниченное количество наблюдателей.

Способы реализации паттерна Observer

Существует два способа реализации шаблона Observer.

Первый способ использует классы Observer и Observable из пакета java.util.

Второй способ использует компонентную модель JavaBeans для регистрации событий в компонентах.

“Observer” применяется в тех случаях, когда система обладает следующими свойствами:

- существует, как минимум, один объект, рассылающий сообщения
- имеется не менее одного получателя сообщений, причём их количество и состав могут изменяться во время работы приложения.

- Нет надобности очень сильно связывать взаимодействующие объекты, что полезно для повторного использования.
- Данный шаблон часто применяют в ситуациях, в которых отправителя сообщений не интересует, что делают получатели с предоставленной им информацией.

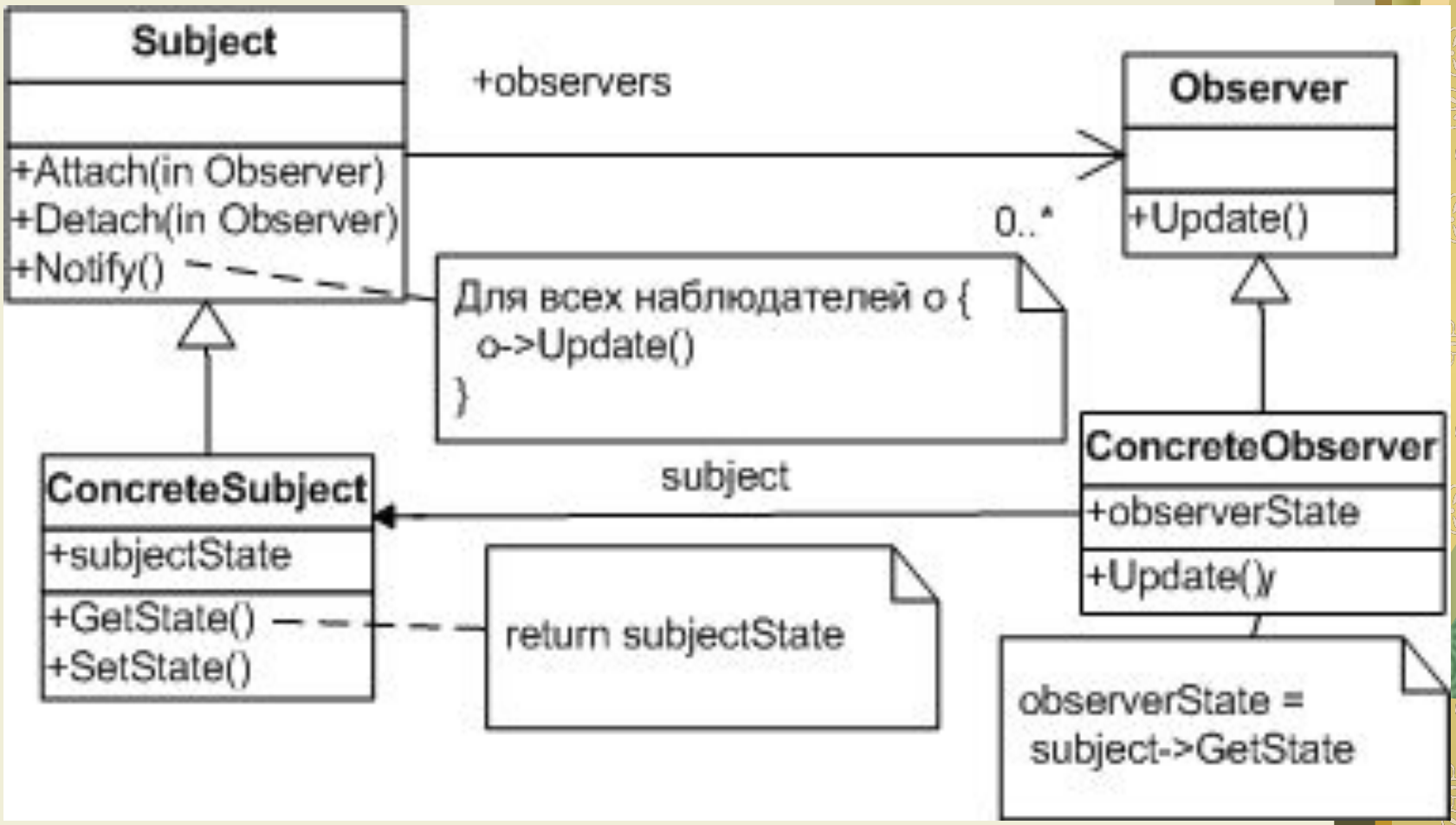
Проблема

Один объект ("Подписчик") должен знать об изменении состояний или некоторых событиях другого объекта. При этом необходимо поддерживать низкий уровень связывания с объектом - "Подписчиком".

Решение

Определить интерфейс "Подписки". Объекты - подписчики реализуют этот интерфейс и динамически регистрируются для получении информации о некотором событии. Затем при реализации условленного события оповещаются все объекты - подписчики.

Принцип действия



Пример использования Observer

Класс Users

```
package observer;

import java.util.ArrayList;
import java.util.List;
import java.util.Observable;

public class Users extends Observable {
private List<String> loggedIn = new ArrayList<String>();

public void login(String login(String userName, String password) {
if (!passwordValid(userName, password)) {
throw new SecurityException("Пользователь с именем " + userName +
" и паролем '" + password + "' - не найден!");
}

loggedIn.add(userName);
setChanged();
notifyObservers(userName);
}

public void logout(String userName) {
loggedIn.remove(userName);
setChanged();
notifyObservers(userName);
}
```

Пример использования Observer

- **public final boolean** passwordValid(String passwordValid(String name, String password) {
- **boolean** res = **false**;
- */* ....
- * *Проверка существования пользователя*
- * *с логином name и паролем password*
- * ....
- * */*
- **return** res;
- }
-
- **public boolean** loggedIn(String name) {
- **return** loggedIn.contains(name);
- }
- }

Пример использования Observer

Класс Eye

```
package observer;

import java.util.Observable;
import java.util.Observer;

public class Eye implements Observer {
Users watching;

public Eye(Users users) {
this.watching = users;
watching.addObserver(this);
}

public void update(Observable users, Object userName) {
if (users != watching) {
throw new IllegalArgumentException();
}

StringString name = (String) userName;
if (watching.loggedIn(name)) {
addUser(name);
} else {
removeUser(name);
}

}
```

Пример использования Observer

```
public void removeUser(String name) {
/* ...
 * Удаляем пользователя из списка
 * присутсвующих
 * ....
 */

}

public void addUser(String name) {
/* ...
 * Регистрируем пользователя в списке
 * присутсвующих
 * ...
 */
}

}
```

Преимущества и недостатки

- Абстрактная связанность субъекта и наблюдателя.
  Субъект имеет информацию лишь о том, что у него есть ряд наблюдателей, каждый из которых подчиняется простому интерфейсу абстрактного класса **Observer**. Субъекту **неизвестны** конкретные классы наблюдателей. Таким образом, связи между субъектами и наблюдателями носят абстрактный характер и сведены к минимуму.
  Поскольку субъект и наблюдатель не являются тесно связанными, то они могут находиться на разных уровнях абстракции системы. Субъект более низкого уровня может уведомлять наблюдателей, находящихся на верхних уровнях, не нарушая иерархии системы. Если бы субъект и наблюдатель представляли собой единое целое, то получающийся объект либо пересекал бы границы уровней (нарушая принцип их формирования), либо должен был находиться на каком-то одном доступном уровне (компрометируя абстракцию уровня).

Преимущества и недостатки

- Поддержка широковещательных коммуникаций.

  В отличие от обычного запроса для уведомления, посылаемого субъектом, не нужно задавать определенного получателя. Уведомление автоматически поступает всем подписавшимся на него объектам. Субъекту не нужна информация о количестве таких объектов, от него требуется всего лишь уведомить своих наблюдателей. Поэтому мы можем в любое время добавлять и удалять наблюдателей.
  Наблюдатель сам решает, обработать полученное уведомление или игнорировать его.

Преимущества и недостатки

- Неожиданные обновления.

Поскольку наблюдатели не располагают информацией друг о друге, им неизвестно и о том, во что обходится изменение субъекта. Безобидная, на первый взгляд, операция над субъектом может вызвать целый ряд обновлений наблюдателей и зависящих от них объектов. Более того, нечетко определенные или плохо поддерживаемые критерии зависимости могут стать причиной непредвиденных обновлений, отследить которые очень сложно.
Эта проблема усугубляется еще и тем, что простой протокол обновления не содержит никаких сведений о том, что именно изменилось в субъекте.
Без дополнительного протокола, помогающего выяснить характер изменений, наблюдатели будут вынуждены проделать сложную работу для косвенного получения такой информации.

Вопросы

- **Что представляет собой паттерн Observer?**

а) паттерн, который предоставляет необходимые функции, но не поддерживает нужного интерфейса

б) паттерн поведения объектов, устанавливающий систему оповещения объектами своих соседей в процессе их деятельности

в) паттерн, контролирующий доступ к объектам, перехватывая все вызовы.

Вопросы

- **Паттерн-наблюдатель определяет зависимость типа?**

а) «один ко многим»

б) «один к одному»

в) «многие ко многим»

- **Один из способов реализации паттерна «Наблюдатель»?**

а) использует классы Observer и Observable

б) использует классы Observer и Runnable

в) использует классы Observer и Iterable