

# Disciplina: Programação III

## Tema: observer, classes anônimas e internas, exceções e ouvintes

---

PROFESSOR: DANILO MONTEIRO

EMAIL: PROF.DANILO.MONTEIRO@GMAIL.COM



# Objetivos da aula

---

Perceber quais são as vantagens e desvantagens na utilização de frameworks orientados a objeto

Compreender como e onde utilizar o Framework **Java Collections**

# Padrão observer

---

## *Definição*

*Permite que um objeto, observado, notifique automaticamente todos os objetos vinculados a ele (objetos observadores) respeitando a relação um-para-muitos. A notificação ocorre assim que o estado do objeto observado é atualizado.*

# Padrão observer

---

O Observer é útil quando precisamos que dois ou mais objetos "escutem" a determinados eventos em um outro objeto. Os objetos que estão escutando são conhecidos como Observers e o objeto que é escutado (ou observado) é conhecido como Subject.

---

# Como assim ?

---

Digamos que você é desenvolvedor em uma empresa, e um dia você descobre que vai precisar faltar a semana inteira. outras pessoas tem que ser avisadas imediatamente:

Seu Gerente de projetos;

- Ele precisa ser avisados para organizar o trabalho dos colegas de equipe e tentar diminuir os efeitos da sua falta

O Rh;

- Precisa colocar falta em você e descontar seu salário.

---

A partir do momento que você não passa o crachá até 8:00 h , todas essas pessoas são avisadas por email de sua falta, e ficam ciente disso.

Ao mesmo tempo todo mundo precisa saber que você chegou, quando você voltar...

Portanto:

o evento é sua falta ou presença na empresa, neste caso, o momento que você passa ou não o crachá até 8h.

o observers são todos os que irão receber a mensagem, eles estão te observando.

o subject é você, que está sendo observado

---

Vamos colocar a mão na massa!

Código disponível em:

<https://github.com/dan1lo/ProgramacaoIII>



---

Primeiro temos que criar um interface observado;

Essa interface vai ter 3 métodos no nosso exemplo, geralmente são os 3;

O método addObserver

- Este método vai adicionar um observador da lista

O método removeObserver

- Este método vai remover um observado da lista

O método notifyObserver

- Este método vai notificar todos os observadores

```
package aulaObserver;

public interface Observado {
    public void addObserver(Observador observer);
    public void removeObserver(Observador observer);
    public void notifyObservers(String p);
}
```

No nosso caso, queremos avisar aos gerente e rh a presença ou a falta de um programador, por isso o método notifyObservers tem como parâmetro um string (que vai ser p = presença e f = a falta)

Os métodos add e remove recebem como parâmetro objetos observadores

```
package aulaObserver;  
  
public interface Observador {  
    public void update(String presenca );  
}
```

---

Agora vamos criar a interface observador

Ela vai ter um método chamado update, que recebe uma string como parametro (que vai ser usada para saber se o cara está presente ou faltou)

---

Vamos criar as classes gerente e rh, elas vão implementar a interface observador( porque elas precisam saber quando o programador faltou ou não)

Como elas são interfaces, ela vão herdar o método da classe observador que a gente acabou de criar (update)

Além disso, nos vamos colocar o comportamento que desejamos no método update quando o gerente e o rh souber se o programador faltou ou tá presente.

```
package aulaObserver;
```

```
public class Gerente implements Observador {
```

```
    // o gerente que está observando, ele precisa ser avisado quando alguém faltar  
    Observado obs;
```

```
    public Gerente(Observado obs) {
```

```
        // construtor, vai pegar o objeto observador, neste caso o programador
```

```
        // e vai adicionar a lista de observadores
```

```
        this.obs = obs;
```

```
        this.obs.addObserver( this );
```

```
    }
```

Nesse momento você não precisa se preocupar com o construtor ( código dentro do retângulo)

```
@Override
```

```
public void update(String presenca) {
```

```
    // TODO Auto-generated method stub
```

```
    // o funcionario mudou de status
```

```
    // vamos considerar que a mudança é falta
```

```
    if (presenca.equals("p")) {
```

```
        System.out.println("Opa funcionario presente! tudo certo");
```

```
    }else {
```

```
        System.out.println("Eita, um funcionario faltou, tenho que rever o cronograma");
```

```
    }
```

```
}
```

```
}
```

```
package aulaObserver;
```

```
public class RH implements Observador{  
    Observado obs;
```

```
    public RH(Observado obs) {  
        // construtor, vai pegar o objeto observador, neste caso o programador  
        // e vai adicionar a lista de observadores  
        this.obs = obs;  
        this.obs.addObserver( this );  
    }
```

Nesse momento você não precisa se preocupar com o construtor ( código dentro do retângulo)

```
@Override
```

```
public void update(String presenca) {  
    //semelhante ao comentário da classe gerente  
    if (presenca.equals("p")) {  
        System.out.println("Opa funcionario presente! vai assinar a folha de ponto");  
    }else {  
        System.out.println("Opa, um funcionário faltou, vou botar falta no ponto");  
    }  
}
```

```
}
```

```
// assim como o gerente, o rh é o observador, ele precisa ser avisado
```

---

Agora vamos para nossa classe que está sendo observada (programador)

Como o método a classe programador é o observado temos que implementar a interface observador e seus métodos

`addObserve()`

`removeObserve()`

`notifyObserve()`

Além disso vamos criar um `listadeobservadores` para que se tenha uma lista de classe que precisam saber quando a atualização existir

```
package aulaObserver;

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

public class Programador implements Observado {
    // o programador é que precisa ser observado
    private List<Observador> listaObservadores; /* NOSSAS ENTIDADES OBSERVADORAS ESTÃO AQUI */

    private String presenca;

    public Programador() {
        //construtor para quando iniciar a classe ter a lista de observadores
        listaObservadores = new ArrayList<>();
    }

    public void setPresenca(String p) {

        this.presenca = p;
        notifyObservers(p); //chamo o metodo notify aqui e passo o status do funcionario
    }

    @Override
    public void addObserver(Observador observer) {
        // adiciono um observador na lista
        listaObservadores.add(observer);
    }
}
```



```
package aulaObserver;

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

public class Programador implements Observado {
    // o programador é que precisa ser observado
    private List<Observador> listaObservadores; /* NOSSAS ENTIDADES OBSERVADORAS ESTÃO AQUI */


    private String presenca;

    public Programador() {
        //construtor para quando iniciar a classe ter a lista de observadores
        listaObservadores = new ArrayList<>();
    }

    public void setPresenca(String p) {

        this.presenca = p;
        notifyObservers(p); //chamo o metodo notify aqui e passo o status do funcionario
    }

    @Override
    public void addObserver(Observador observer) {
        // adiciono um observador na lista
        listaObservadores.add(observer);
    }
}
```



```
@Override
public void removeObserver(Observador observer) {
    // aqui eu vou remover algum observador da lista caso eu não deseje ele mais observando
    // tenho que verificar se ele tá ou na dentro da lista
    int index = listaObservadores.indexOf( observer );
    if( index > -1 ){
        listaObservadores.remove( observer );
    }
}
```

```
@Override
public void notifyObservers(String p) {

    // aqui é onde toda a mágica acontece
    // a gente tem uma lista de observadores
    // todos os observadores tem um método update
    // como todos os observadores estão dentro da lista
    // este metodo, ao ser chamado, executa todos os metodos updates em todas as classes que temos observando

    for( Observador o :listaObservadores ){
        o.update(p);
    }
}
```

---

Agora vamos criar um construtor para o programador que instancia a lista toda vez que o programador é criado

(lembre-se de criar o atributo presença também na classe programador 😊 )

```
package aulaObserver;

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

public class Programador implements Observado {
    // o programador é que precisa ser observado
    private List<Observador> listaObservadores; /* NOSSAS ENTIDADES OBSERVADORAS ESTÃO AQUI */


    private String presenca;

    public Programador() {
        //construtor para quando iniciar a classe ter a lista de observadores
        listaObservadores = new ArrayList<>();
    }

    public void setPresenca(String p) {

        this.presenca = p;
        notifyObservers(p); //chamo o metodo notify aqui e passo o status do funcionario
    }

    @Override
    public void addObserver(Observador observer) {
        // adiciono um observador na lista
        listaObservadores.add(observer);
    }
}
```



---

Vamos criar o método `setPresenca()` para alterar se o cara faltou ou não

Note que neste método é o evento que queremos monitorar, ou seja, todas as outras classe vão observar mudanças nesse método, e por isso que ele vai ter uma chamada para outro método na nossa classe chamado de `notifyObserver()`

```
package aulaObserver;

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

public class Programador implements Observado {
    // o programador é que precisa ser observado
    private List<Observador> listaObservadores; /* NOSSAS ENTIDADES OBSERVADORAS ESTÃO AQUI */


    private String presenca;

    public Programador() {
        //construtor para quando iniciar a classe ter a lista de observadores
        listaObservadores = new ArrayList<>();
    }

    public void setPresenca(String p) {

        this.presenca = p;
        notifyObservers(p); //chamo o metodo notify aqui e passo o status do funcionario
    }

    @Override
    public void addObserver(Observador observer) {
        // adiciono um observador na lista
        listaObservadores.add(observer);
    }
}
```



---

Agora vamos voltar para as classes gerente e rh e adicionar no construtor algumas modificações necessárias

```
package aulaObserver;
```

```
public class Gerente implements Observador {  
    // o gerente que está observando, ele precisa ser avisado quando alguém faltar  
    Observado obs;
```

```
    public Gerente(Observado obs) {  
        // construtor, vai pegar o objeto observador, neste caso o programador  
        // e vai adicionar a lista de observadores  
        this.obs = obs;  
        this.obs.addObserver( this );  
    }
```

Eu preciso receber quem é o observado, e passar para a lista do observado que esta classe é um observador  
Com o método addObserver passando o this

```
    @Override  
    public void update(String presenca) {  
        // TODO Auto-generated method stub  
        // o funcionario mudou de status  
        // vamos considerar que a mudança é falta  
        if (presenca.equals("p")) {  
            System.out.println("Opa funcionario presente! tudo certo");  
        } else {  
            System.out.println("Eita, um funcionario faltou, tenho que rever o cronograma");  
        }  
    }  
}
```

```
}
```



```
package aulaObserver;
```

```
public class RH implements Observador{  
    Observado obs;
```

```
    public RH(Observado obs) {  
        // construtor, vai pegar o objeto observador, neste caso o programador  
        // e vai adicionar a lista de observadores  
        this.obs = obs;  
        this.obs.addObserver( this );  
    }
```

Eu preciso receber quem é o observado, e passar para a lista do observado que esta classe é um observador  
Com o método addObserver passando o this

```
    @Override  
    public void update(String presenca) {  
        //semelhante ao comentário da classe gerente  
        if (presenca.equals("p")) {  
            System.out.println("Opa funcionario presente! vai assinar a folha de ponto");  
        }else {  
            System.out.println("Opa, um funcionário faltou, vou botar falta no ponto");  
        }  
    }
```

```
}  
// assim como o gerente, o rh é o observador, ele precisa ser avisado
```

```
package aulaobserver;

public class Principal {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Programador p = new Programador();
        Gerente g = new Gerente(p);
        RH rh = new RH(p);

        System.out.println("vamos colocar presença no programador");
        p.setPresenca("p");
        System.out.println("vamos colocar falta no programador");
        p.setPresenca("f");
    }

}
```

Agora vamos executar nossos comandos em um main

Eu preciso criar um programador

Eu preciso criar um gerente e passar como parâmetro o programador

Eu preciso criar um rh e passar como parâmetro o programador

---

Faça o teste 😊

# Referencias

---

<https://www.thiengo.com.br/padrao-de-projeto-observer>

<https://brizenowordpress.wordpress.com/category/padroes-de-projeto-observer/>

