EARLY POO

class and package design

Contreras Huamani, Paul Michaell
Flores Silva, David

[1]System Engineering School
System Engineering and Informatic Department
Production and Services Faculty
San Agustin National University of Arequipa

2020-04-01

Content

Class design

symptoms of a class design

- Rigidity.
- Fragidity.
- Inmovility.
- Viscosity.
- Innesesary Complexity.
- Innesesary Repetition.
- Opacity.

1.- Rigidity
classes are difficult to change
2.- Fragidity
classes stop working
3.- Inmovility
classes are difficult to reuse
4.- Viscosity
classes are difficult to use
5.- Innesesary Complexity
overdesigned
6.- Innesesary Repetition
copy and paste
7.- Opacity
messy

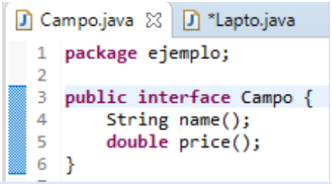
Principles of class designs

Eliminate the symptoms of a Class Design

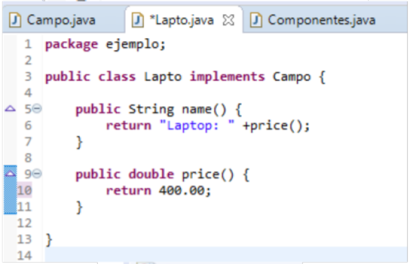
- Sole responsability.
- Open and closed.
- Liskov substitution.
- Independence investment.
- Interface segregation.

1.- Sole responsability
only reason to change
2.- Open and closed
open extended and closed modify
3.- Liskov subtitution
is replaced by subtypes...
4.- Independence investment
details depend on abstractions
5.- Interface segregation
a class does not depend on interfaces that do not use

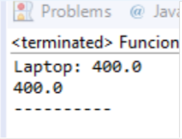
BASIC EXAMPLE



```java
package ejemplo;

public interface Campo {
      String name();
      double price();
}
```

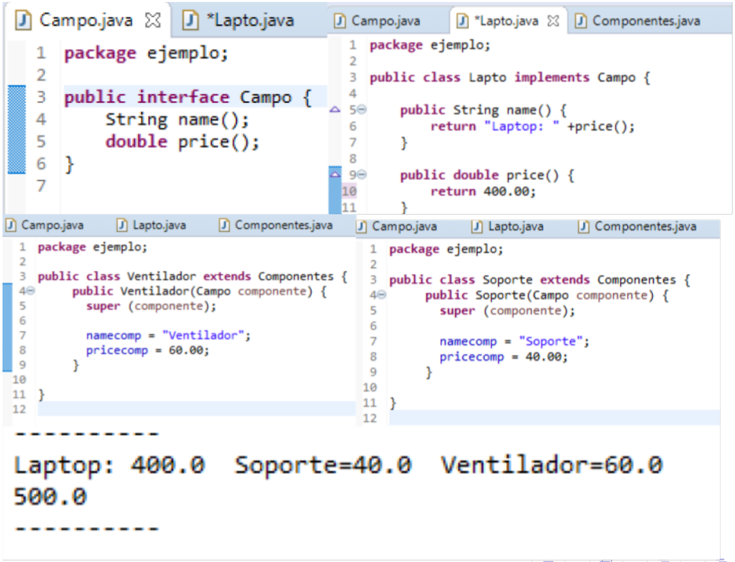


```java
package ejemplo;

public class Lapto implements Campo {

    public String name() {
        return "Laptop: " +price();
    }

    public double price() {
        return 400.00;
    }

}
```



```
<terminated> Funcion
Laptop: 400.0
400.0
----------
```

ADVANCED EXAMPLE



Campo.java

```java
package ejemplo;

public interface Campo {
    String name();
    double price();
}

```

*Lapto.java

```java
package ejemplo;

public class Lapto implements Campo {

    public String name() {
        return "Laptop: " +price();
    }

    public double price() {
        return 400.00;
    }
```

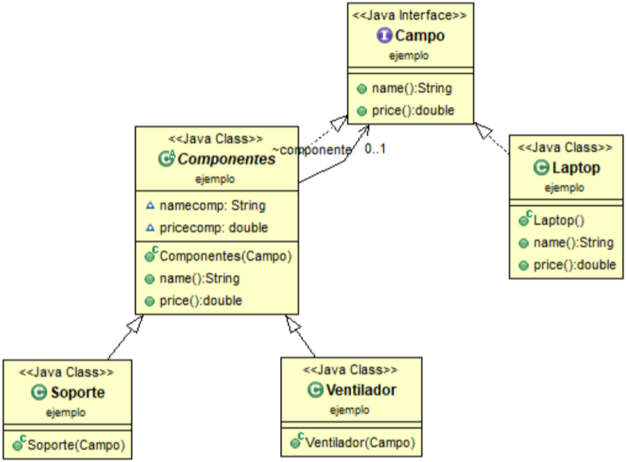
Ventilador

```java
package ejemplo;

public class Ventilador extends Componentes {
    public Ventilador(Campo componente) {
      super (componente);

      namecomp = "Ventilador";
      pricecomp = 60.00;
    }

}

```

Soporte

```java
package ejemplo;

public class Soporte extends Componentes {
    public Soporte(Campo componente) {
      super (componente);

      namecomp = "Soporte";
      pricecomp = 40.00;
    }

}

```

```
-----------
Laptop: 400.0  Soporte=40.0  Ventilador=60.0
500.0
-----------
```



<<Java Interface>>
Campo
ejemplo
name():String
price():double
<<Java Class>>
Componentes
ejemplo
namecomp: String
pricecomp: double
Componentes(Campo)
name():String
price():double
~componente
0..1
<<Java Class>>
Laptop
ejemplo
Laptop()
name():String
price():double
<<Java Class>>
Soporte
ejemplo
Soporte(Campo)
<<Java Class>>
Ventilador
ejemplo
Ventilador(Campo)

Classes abstract and interface

An abstract class has two main functions which are:

- has no instance.
- subclasses are defined.

And an interface is declared with the keyword **INTERFACE** and it works

- method statements.
- cannot instantiate.
- appear on packages.

Package design
Cohesion and coupling

- Granularity: The cohesion of the packages
- Stability: The coupling between packages

Case study: Mobile phone network

Operation of a mobile telephone network

Elements :

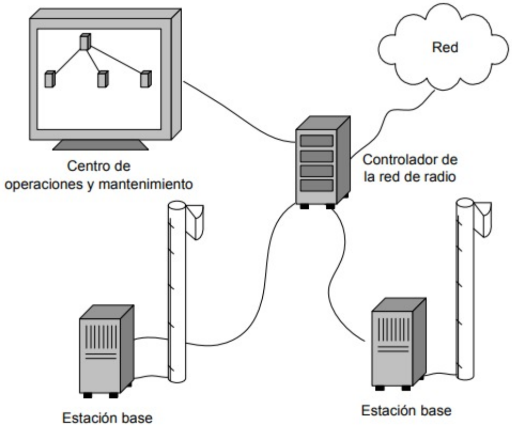
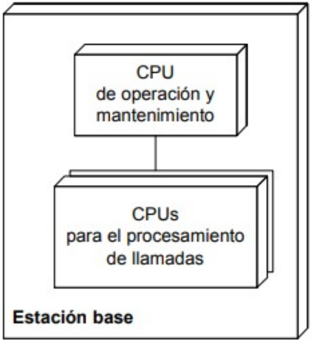


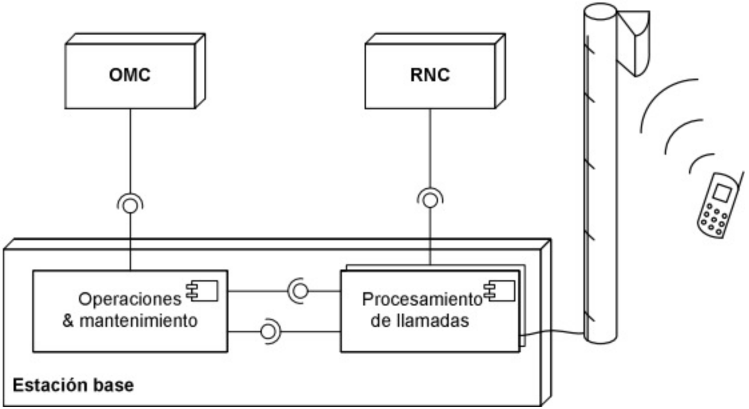
Figura original cortesía de Michael Kircher

The architecture of a base station

1.- Base station hardware



2.- Base station software

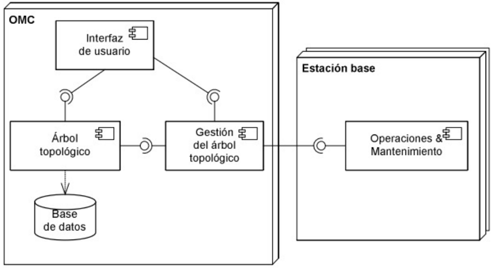


OMC software architecture

1.- Functions of OMC

- Communication with base stations and RNCs.
- Discovery of base stations.
- Maintaining network status.
- Configuration of network elements.
- Base Station Software Update.

2.- OMC software architecture

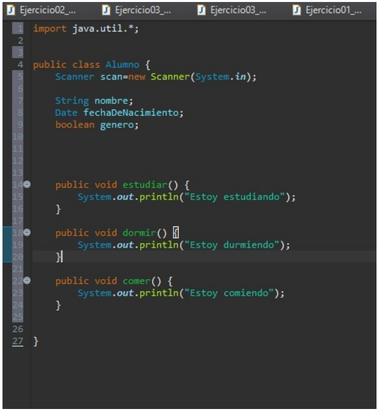


El patrón de diseño MVC
(Model-View-Controller = Modelo-Vista-Controlador)

3.- The OMC software is organized according to the MVC pattern

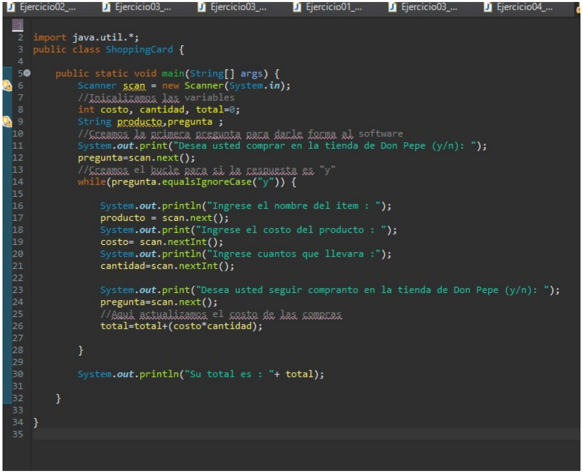
- Model.
- view.
- Controller.

BASIC EXAMPLE 2



```
import java.util.*;

public class Alumno {
    Scanner scan=new Scanner(System.in);

    String nombre;
    Date fechaDeNacimiento;
    boolean genero;




    public void estudiar() {
        System.out.println("Estoy estudiando");
    }

    public void dormir() {
        System.out.println("Estoy durmiendo");
    }

    public void comer() {
        System.out.println("Estoy comiendo");
    }

}
```

ADVANCED EXAMPLE 2



```java

import java.util.*;
public class ShoppingCard {

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        //Inicalizamos las variables
        int costo, cantidad, total=0;
        String producto,pregunta ;
        //Creamos la primera pregunta para darle forma al software
        System.out.print("Desea usted comprar en la tienda de Don Pepe (y/n): ");
        pregunta=scan.next();
        //Creamos el bucle para si la respuesta es "y"
        while(pregunta.equalsIgnoreCase("y")) {

            System.out.println("Ingrese el nombre del item : ");
            producto = scan.next();
            System.out.print("Ingrese el costo del producto : ");
            costo= scan.nextInt();
            System.out.println("Ingrese cuantos que llevara :");
            cantidad=scan.nextInt();

            System.out.print("Desea usted seguir compranto en la tienda de Don Pepe (y/n): ");
            pregunta=scan.next();
            //Aqui actualizamos el costo de las compras
            total=total+(costo*cantidad);

        }

        System.out.println("Su total es : "+ total);

    }

}

```

BIBLIOGRAPHY

1. https://elvex.ugr.es/decsai/java/
2. https://www.instintoprogramador.com.mx/2020/07/principio-abierto-cerrado.html
3. https://desarrolloweb.com/articulos/principio-sustitucion-liskov-dotnet.html
4. https://elvex.ugr.es/decsai/java/pdf/AB-classes.pdf
5. https://elvex.ugr.es/decsai/java/pdf/AD-packages.pdf
6. https://elvex.ugr.es/decsai/java/pdf/AF-ejercicios.pdf