SOLID HAUTAZKO LABORATEGIA (Ander Pollacino eta Julen Larrañaga) GITHUB

Ireki-itxia printzipioa (OCP)

Zer gertatuko litzateke Sheet klaseari "Diamond" gehituko bagenio? Sheet klaseak OCP printzipioa betetzen du?

Diamond klasea gehitu eta hori marraztu ahal izateko, sheet klasean Diamond objetuak gordetzen dituen bektore berri bat sortu, bektorean elementuak sartzeko addDiamond metodo berria idatzi, eta gainera drawFigures metodoan beste begizta bat sortu beharko genuke.

Begibistakoa da OCP printzipioa ez duela betetzen, forma edo irudi mota berri bat txertatzea erabakitzen dugun bakoitzean, objektu berri horren klase berria sortzeaz aparte, sheet klasean ere kodea aldatu beharko dugudalako, eta gainera kode bera edo oso antzeko behin eta berriro errepikatuz.

Sheet klaseak OCP bete ahal izateko, figureren umeak erabili beharrean, figure superklasea erabil dezan aldatu beharko dugu:

- Bektore bakarra Figure motako objetuak(eta bere umeak) gordetzen dituena. Square eta circle bektoreak ezabatu.

- addFigure metodoa bektore horretara objetuak sartzeko. AddSquare eta addCircle metodoak ezabatu.

- DrawFigure eraldatu, bere barruan begizta bakarra egonik, figure bektoreko elementu guztiak atzituz eta draw eginez.

```
public class Sheet {

    private Vector<Figure> figures = new Vector<Figure>();

    public void addFigure(Figure f){
        figures.add(f);
    }

    public void drawFigures(){
        Enumeration<Figure> efigure = figures.elements();
        Figure s;
        while (efigure.hasMoreElements()){
            s=efigure.nextElement();
            s.draw();
        }
    }
```

Aplikazio bat sortu behar diren klase guztiekin, eta programa nagusi batean, drawFigures() metodoari deitzen dion programa bat sortu, bi circle, square bat eta diamond objektu batekin.

```
public static void main(String[] args) {

    Sheet sheet = new Sheet();
    Circle circle = new Circle((float) 1.5);
    Circle circle1 = new Circle((float) 2.5);
    Square square = new Square((float) 0.3);
    Diamond diamond = new Diamond((float) 0.3, (float) 0.3);

    sheet.addFigure(circle);
    sheet.addFigure(circle1);
    sheet.addFigure(square);
    sheet.addFigure(diamond);

    sheet.drawFigures();
    System.out.println("--------------------------------");
```

Egin duzun aldaketa errefaktorizazio bat dela kontsideratzen duzu?

Errefaktorizazioaren arabera, aurretik egindako kode batean errefaktorizazio bat egitean, kodeak lehen bezala funtzionatzen segi beharko luke.

Kasu honetan, adibidez, kanpoko metodo edo kode zati batek sheet klaseko addSquare edo addCircle metodoetako bateri deitzen bazioen, errefaktorizazioan metodo horiek ezabatu ditugunez, erroreak suertatuko dira. Beraz, ez da errefaktorizazio bat.

Figure bakoitzak azalera bat edukiko balu (hau da, getArea()), eta Sheet batean dauden irudi guztien azalera kalkulatu nahi bagenu, nola osatuko zenuke klaseak.

Alde batetik, figure klasean getArea metodo abstraktua definituko nuke, bere ume guztiek inplementa zezaten. Bestalde, Sheet klasean gordetako figure objetuen getArea metodoei deitu beharko genieke, getFigureAreas izeneko metodo berri bat sortuz adibidez.

Erresponsabilitate bakarreko printzipioa (SRP)

Aplikazioa errefaktoriazatu, erresponsabilitate bakoitza klase isolatu batean geratu dadin.

```java
public class BillDeduction {

    private float billAmount;
    private int deductionPercentage;

    public BillDeduction(float billAmount, int deductionPercentage) {
        this.billAmount = billAmount;
        this.deductionPercentage = deductionPercentage;
    }

    public float getBillAmount() {
        return billAmount;
    }

    public void setBillAmount(float billAmount) {
        this.billAmount = billAmount;
    }

    public int getDeductionPercentage() {
        return deductionPercentage;
    }

    public void setDeductionPercentage(int deductionPercentage) {
        this.deductionPercentage = deductionPercentage;
    }

    public float calculateBillDeduction() {
        return (billAmount * deductionPercentage) / 100;
    }
```

```java
  public class Vat {

        private double value = 0.16;
        private float billAmount;

        public Vat(float billAmount) {
            this.billAmount = billAmount;
        }

        public double getValue() {
            return value;
        }

        public void setVat(double value) {
            this.value = value;
        }

        public float getBillAmount() {
            return billAmount;
        }

        public void setBillAmount(float billAmount) {
            this.billAmount = billAmount;
        }

        public float calculateVAT() {
            return (float) (billAmount * value);
        }
```

```java
public class Bill {

    public String code;
    public Date billDate;
    public float billAmount;
    public float billDeduction;
    public float billTotal;
    public float VAT;
    public int deductionPercentage;

    // Fakturaren totala kalkulatzen duen metodoa.
    public void totalCalc() {

        // Dedukzioa kalkulatu
        BillDeduction cbd = new BillDeduction(billAmount, deductionPercentage);
        billDeduction = cbd.calculateBillDeduction();

        // VAT kalkulatzen dugu
        Vat cvat = new Vat(billAmount);
        VAT = cvat.calculateVAT();

        // Totala kalkulatzen dugu
        billTotal = billAmount - billDeduction + VAT;
    }
```

Adierazi zein aldaketa egin beharko litzateke fakturaren dedukzioa (hau da, billDeduction) fakturaren arabera kalkulatuko balitz.

BillDeduction klasea denez billDeduction kalkulatzeaz arduratzen dena, kode aldaketa hori klase horretako calculateBillDeduction metodoan bakarrik egin beharko genuke.

Zein aldaketa egin beharko litzateke VAT-a %16-tik %18-ra aldatuko balitz?

Vat klaseko vat atributua aldatu beharko genuke. Horretarako vat.setValue(0.18) besterik ez genuke egin beharko.

Zein aldaketa egin beharko litzateke, 0 kodea duten fakturei VAT-a aplikatzen EZ bazaie.

Kodea Bill klaseko atributua denez, klase horretako totalCalc metodoa aldatu beharko genuke, gakoa 0 deneko kasu hori kontuan izan dezan.

Liskov printzipioa (LSK)

Project klaseak OCP printzipioa betetzen du?

Bai, sortzen dugun kode berria ProjectFile klasearen ume den bitartean. Project klasean gordetzen ditugun objetuak ProjectFile motakoak direlako.

Project klaseak LSK printzipioa betetzen du?

Ez du betetzen, ProjectFile klasearen ReadOnlyProjectFile klase umeak ezin duelako superklasetik eratorritako storeFile metodoa inplementatu.

Errefaktorizatu aplikazioa OCP edo LSK betetzen ez badu.

ProjectFile superklasetik eratortzen diren umeak 3 motakoak izan daitezke: irakurtzeko soilik, idazteko soilik, edo irakurri eta idazteko. Hori adierazteko 3 interfaze adierazi ditugu, ProjectFile klasea aldatzeaz aparte:

```
public interface LoadableFile{

    public void loadFile();

}
```

```
public interface StorableFile{

    public void storeFile();

}
```

```
public interface StorableAndReadable extends LoadableFile, StorableFile{

}
```

```
public abstract class ProjectFile {
    public String filePath;

    public ProjectFile(String filePath){
        this.filePath=filePath;
    }

}
```

Beraz, ReadOnlyProjectFile klaseak, ProjectFile klasearen ume izango da eta gainera, LoadableFile interfazea inplementatuko du:

```
public class ReadOnlyProjectFile extends ProjectFile implements LoadableFile{

    public ReadOnlyProjectFile(String filePath) {
        super(filePath);
    }

    public void loadFile(){
        System.out.println("file loaded from "+filePath);
    }

}
```

Bukatzeko, Project klasea ere aldatu behar dugu, alde batetik irakurtzeko bakarrik direnak gordetzeko, edo bestetik idazteko (bi aukerak inplementatzen dituztenak bi listetan azalduko dira):

```
public class Project {

    public Vector<StorableFile> sfiles = new Vector<>();
    public Vector<LoadableFile> lfiles = new Vector<>();


    public void addLoadableProject(LoadableFile p){
        lfiles.add(p);
    }
    public void addStorabeProject(StorableFile p){
        sfiles.add(p);
    }
    public void addStorabeAndReadableProject(StorableAndReadable p){
        sfiles.add(p);
        lfiles.add(p);
    }

    public void loadAllFiles(){
        for (LoadableFile lf:lfiles)
            lf.loadFile();
    }

    public void storeAllFiles(){
        for (StorableFile sf:sfiles)
            sf.storeFile();
    }
}
```

Dependentzi inbertsioaren printzipioa (DIP)

DIP printzipioa betetzen du? Justifikatu erantzuna.

Ez du printzipioa betetzen, register metodoa, beste klaseak zuzenean sortu eta beraien implementazioarekiko mempekoa delako.

Adibidez aurrebaldintzak konprobatzeko klasea aldatuko bagenu, register metodoan erroreak suertatuko litzateke eta eskuz aldatu beharko genuke kodea.

Gainera, register metodoarentzako testak inplementatu nahiko bagenitu, metodoa beste klaseekiko isolatzea zaila izango litzateke.

Errefaktorizatu kodea betetzen ez badu.

Kodea zuzentzako, alde batetik, klase zehatzekiko menmekoa izan beharrean, interfazeak erabil ditzan egingo dugu. Horretarako menpekotasun bakoitzeko interfaze bat sortuko dugu.

Aurrebaldintzak konprobatzeko, aurretik erabiltzen dugun Precondition klaseak, irudian erakusten dugun PreconditionChecker interfazea inplementatuko luke:

```
public interface PreconditionChecker {

    public boolean isPossible(String subject, HashMap<String,Integer> subjectRecord);

}
```

Dedukzioa kalkulatzeko, aurretik erabiltzen dugun Deduction klaseak, irudian erakusten dugun DeductionCalculator interfazea implementatuko luke:

```
public interface DeductionCalculator {

    public int calcDeduction(String sex, String year);

}
```

Irakasgaiaren prezioa kalkulatzeko, aurretik erabiltzen dugun SubjectQuotes klaseak, irudiko SubjectPriceCalculator interfazea inplementatuko luke:

```java
public interface SubjectPriceCalculator {

    public int getPrice(String subject);

}
```

Bukatzeko, aldaketa horiek egin ondoren, register klasean instantziak sortu beharrean, interfaze konkretuak inplementatzen dituzten objetuak pasako dizkiogu parametro moduan, horiek inplementatzen dituzten metodoak erabil ditzan:

```java
public class Student {
    public String name;
    public String sex;
    public String year;
    // matrikulatuta dagoen espedientea
    public HashMap<String,Integer> subjectRecord;
    // ikasleak ordaindu behar duena
    public String toCharge;

    // Irakasgai batean matrikulatzen duen metodoa.
    public void register(String subject, PreconditionChecker preCheck, DeductionCalculator deductCalc, SubjectPriceCalculator subjPriceCalc) {
        // Aurrebaldintzak konprobatzen dira
        boolean isPosible = preCheck.isPosible(subject , subjectRecord);
        if (isPosible) {
            // Dedukzioa kalkulatu sex eta edadearen arabera
            int percentage = deductCalc.calcDeduction(sex, year);
            // Irakasgaiaren prezioa lortu
            int quote = subjPriceCalc.getPrice(subject);
            // HashMap batean gordetzen du eta ordaindu behar duen balioa eguneratu
            subjectRecord.put(subject,null);
            toCharge = toCharge+(quote-percentage*quote/100);
        }
    }
}
```

Interfaze Bananduaren Printzipioa (ISP)

Zer informazio behar dute EmailSender eta SMSSender klaseek egin behar duten funtzionalitatea betetzeko, eta zer informazio jasotzen dute? ISP printzipioa bortxatzen dutela uste duzu?

Bi klase horiek adierazitako testua nori bidali jakiteko, pertsonaren posta helbidea edo telefonoa bakarrik behar dute. Aldiz, Pertsona objetua pasatzen zahie parametro moduan, helbidea edo telefonoa bakarrik pasa beharrean.

Hori egitean, ISP printzipioa ez da betetzen, sendEmail eta sendSMS metodoei Person objetua pasatzean, ezin izango dugulako metodo hori berrerabili Pertsona motakoak ez diren objetuentzako, adibidez enpresa bateko posta elektronikora mezu bat bidali ahal izateko.

Aurreko klaseak errefaktorizatu (EmailSender eta SMSSender), Person parametroa aldatuz interface batengatik (klase bakoitzak interfaze desberdina). Interface bakoitzak klase bakoitzak behar duen metodoez osatuta egongo da bere eginkizuna betetzeko, hau da, mail bat edo sms bat bidaltzeko. Person klasea ere aldatu.

Bi interfaze sortu ditugu: HasTelephone izenekoa zein telefonoa duten klaseek inplementatu beharko duten, eta HasEmail izenekoa zein posta elektronikoa duten klaseek inplementatu beharko duten.

```
public interface HasTelephone {

    public void setTelephone(String t);
    public String getTelephone();

}
```

```
public interface HasEmail {

    public void setEmail (String e);
    public String getEmail();

}
```

Behin interfaze horiek sortuta, Person klaseak bai telefonoa, bati posta elektronikoa duenez, HasTelephone eta HasEmail implementa ditzan egin dugu:

```
public class Person implements HasTelephone, HasEmail{

    private String name, address, email, telephone;

    public void setEmail (String e) { email=e; }
    public String getEmail () { return email; }

    public void setTelephone(String t) { telephone=t; }
    public String getTelephone() { return telephone; }
```

Azkenik, EmailSender eta SMSSender klaseak aldatu ditugu, HasEmail edo HasTelephone interfazeak inplementatzen dituzten objetuak jaso ahal izateko, Person motako objetuak bakarrik jaso beharrean:

```
public static void sendEmail(HasEmail c, String message){
    // Mezu bat bidaltzen du Person klaseko korreo helbidera.
}

public static void sendSMS(HasTelephone c, String message){
    //SMS bat bidaltzen du Person klaseko telefono zenbakira.
}
```

Send metodo horietan, jasotako objetuak inplementatuta izan beharko duen geterrari deitu beharko litzateke, kasu bakoitzean lortu nahi den balioaren arabera.

GmailAccount klasea osatu. Klase honetako objektuek (aldaketa batzuekin) EmailSender klasera bidaltzeko ahalmena izan beharko lukete baina ez SMSSender klasera.

```
public class GmailAccount implements HasEmail{

    public String name, emailAddress;

    @Override
    public void setEmail(String e) {
        this.emailAddress = e;

    }

    @Override
    public String getEmail() {
        return this.emailAddress;
    }

}
```

Programa bat sortu, EmailSender klaseko sendEmail metodoari deitzen diona GmailAccount objektu batekin.

```
public class Main {
    public static void main(String[] args) {

        GmailAccount ga = new GmailAccount();
        ga.setEmail("g@gmail.com");

        EmailSender.sendEmail(ga, "probaaaa1");
        SMSSender.sendSMS(ga, "probaaaa2");

    }
}
```

```
oblems  @ Javadoc  Declaration  Console  SonarLint On-The-Fly  Coverage  Git Staging  SonarLint Report
inated> Main (3) [Java Application] C:\Program Files\Java\jdk-11.0.11\bin\javaw.exe  (30 oct. 2021 10:02:54 – 10:02:55)
ption in thread "main" java.lang.Error: Unresolved compilation problem:
    The method sendSMS(HasTelephone, String) in the type SMSSender is not applicable for the arguments (GmailAccount, String)

    at isp.Main.main(Main.java:10)
```