ERREFAKTORIZAZIOA

ERREFAKTORIZAZIOA **1**

Sarrera **1**

Write Short Units of Code **2**
Aurretiko Kodeketa 2
Errefaktorizatutako Kodea 3
Errefaktorizazio Prozedura 4

Keep Small Interfaces Unit **6**
Aurretiko Kodeketa 6
Errefaktorizazioa burutu osteko kodea 7
Errefaktorizazio Prozedura 8

Sarrera

Gu Unax Lazkanotegi eta Oier Elola gara eta dokumentu honetan aurkituko duzuena da gure Bets21 proiektuan egin ditugun aldaketak errefaktorizazio bidez. Hemen agertzen diren bi aldaketak *Building Maintainable Software* liburuan agertzen diren kapituluetan daude oinarritua, zehazki 2. eta 5. kapituluak dira landu direnak (Unaxen kasuan 2. kapitulua izan da landua eta Oierrenean 5.) . Liburua Interneten eskuragarri dago edozeinentzako irakurri dezan.

Errefaktorizazioaren helburu nagusia kodearen mantenua etorkizunera begira errazagoa izatea da; horrela, *code smell* deritzon akatsak burutzea ekidin nahi da. Hainbat errefaktorizazio mota ezberdindu daitezke, baina gure kasuan aipatuko direnak pare bat izango dira.

Beraz, informazio gehiegirik eman gabe, goazen bakoitzak egin duenarekin.

Write Short Units of Code

Errefaktorizazio honen bidez lortuko duguna, erabiltzen ditugun metodoak gizakiarentzat ulergarriagoak izatea da. Hau da, esan ohi dena da edozein zorok dakiela nola programatu ordenagailuak ulertu dezakeen programa bat, baina bakarrik programatzaile onek dakite gizakiak erraz ulertu dezakeen programa bat.

Horregatik, errefaktorizazio honetan planteatzen dena metodoen lerro kopurua limitatzea da, hau da, ikusi berri dugun metodo batek 5 lerro baditu, errazago egingo zaigu ulertzea zer gauza eta nola egiten duen, 50 lerro dituen metodo batean baino.

Idei honetan, guk izango dugun muga, 15 lerrokoa izango da. Noski, erabili behar dugun metodoa jada konplexuagoa bada, nahi eta nahi ez idatzi beharko ditugu 15 lerro baina gehiago, baina hori ez gertatzeko, berak erabiliko dituen beste submetodo batzuetaz baliatuko gara, noski, horiek baldintza berdina betetzen duten heinean.

Aurretiko Kodeketa

Errefaktorizazio honetan, DataAccess klasean daukagun ezabatuGertaera metodoa zuzentzea erabaki dugu, esan bezala, metodo berriak sortuz, kodea ordezteko. Hau zen hasieran genuen metodoaren kodea, errefaktorizatu aurretik:

```
1   public void ezabatuGertaera(Event event) {
2     Event e = db.find(Event.class, event.getEventNumber());
3     db.getTransaction().begin();
4     int x;
5     for (Question question : e.getQuestions()) {
6       for (Pronostikoa pronostic : question.getPronostics()) {
7         for (Apustua bet : pronostic.getApustuak()) {
8           x = bet.removePronostikoa(pronostic);
9           Bezeroa bezeroa=bet.getBezeroa();
10          double komisioa = 0;
11          if(x==0) {
12            if (bet.getErrepikatua()!=null) {
13              Bezeroa erre=bet.getErrepikatua();
14              Errepikapena er=bezeroa.getErrepikapena(erre);
15              er.eguneratuHilHonetanGeratzenDena(bet.getKopurua());
16            }
17            bezeroa.addMugimendua("Apustuaren dirua itzuli
18                          ("+bet.getIdentifikadorea()+")",bet.getKopurua(),"bueltatu");
19            bezeroa.removeApustua(bet);
20            db.remove(bet);
21          }else{
22            if (bet.getErrepikatua()!=null) {
23              Bezeroa bez = bet.getErrepikatua();
24              Errepikapena errepikapen=bezeroa.getErrepikapena(bez);
25              komisioa=(bet.getKopurua()*bet.getKuotaTotala() - bet.getKopurua())
26                        * errepikapen.getKomisioa();
27              bez.addMugimendua("Apustu errepikatuaren komisioa ("+bezeroa+")",
28                                komisioa,"irabazi");
```

```
29            }
30            double irabazia=bet.getKopurua()*bet.getKuotaTotala()-komisioa;
31            bezeroa.addMugimendua("Apustua irabazi ("+bet.getIdentifikadorea()+")",
32            irabazia, "irabazi");
33          }
34        }
35      }
36    }
37    db.remove(e);
38    db.getTransaction().commit();
39  }
```

Ikusi dezakegun moduan, 39 lerro dauzka eta begibistakoa da muga gainditzen duela. Horregatik ideia ona izan da hau errefaktorizatzea, hobeto ulertzeko, eta honela geratu da kodea.

Errefaktorizatutako Kodea

Metodoa berez, honela ikusiko genuke:

```
1  public void ezabatuGertaera(Event event) {
2    Event e = db.find(Event.class, event.getEventNumber());
3    db.getTransaction().begin();
4    borratuGertaerarenApustuak(e);
5    db.remove(e);
6    db.getTransaction().commit();
7  }
```

Bistan denez, hau bakarrik kodetuta ezinezkoa da beste kodeak egiten duen berdina egitea, kontuan izan behar dugu sortu berri den borratuGertaerarenApustuak metodoa. Honen barruan, metodoak zituen gainerako agindu guztiak daude…

```
1   private void borratuGertaerarenApustuak(Event e) {
2     int x;
3     for (Question question : e.getQuestions()) {
4       for (Pronostikoa pronostic : question.getPronostics()) {
5         for (Apustua bet : pronostic.getApustuak()) {
6           x = bet.removePronostikoa(pronostic);
7           pronostikoKopBaldintza(x, bet);
8         }
9       }
10    }
11  }
```

Gainera, ikusi dezakegu behin eta berriro beste metodo berriren bat dagoela kodea ordezkatzen, hau uneoro arrazoi berdinarengandik da, berez metodoak 39 lerro zituen eta. Hurrengo metodoan, pronostikoKopBaldintza-n, pronostiko kopuruaren arabera portaera aldatuko da, hortik dator metodoaren izena eta behin eta berriro berdina egingo du, kode guztia adierazi arte.

```
1    private void pronostikoKopBaldintza(int x, Apustua bet) {
2      Bezeroa bezeroa=bet.getBezeroa();
```

```
3      double komisioa = 0;
4      if(x==0) {
6         etendakoErrepikapena(bet, bezeroa);
7         bezeroa.addMugimendua("Apustuaren dirua itzuli ("+bet.getIdentifikadorea()+")",
                          bet.getKopurua(),"bueltatu");
8         bezeroa.removeApustua(bet);
9         db.remove(bet);
10       }else{
11         komisioa = komisioaOrdaindu(bet, bezeroa, komisioa);
12         double irabazia=bet.getKopurua()*bet.getKuotaTotala()-komisioa;
13         bezeroa.addMugimendua("Apustua irabazi ("+bet.getIdentifikadorea()+")",irabazia,
                             "irabazi");
14       }
15     }
```

Azkenik, metodo honek kasu bakoitzerako beste metodo bat izango du: `etendakoErrepikapena` eta `komisioaOrdaindu`. Hauek azkenekoak izango dira.

```
1  private double komisioaOrdaindu(Apustua bet, Bezeroa bezeroa, double komisioa) {
2    if (bet.getErrepikatua()!=null) {
3      Bezeroa bez = bet.getErrepikatua();
4      Errepikapena errepikapen=bezeroa.getErrepikapena(bez);
5      komisioa=(bet.getKopurua()*bet.getKuotaTotala()-bet.getKopurua())
                 *errepikapen.getKomisioa();
6      bez.addMugimendua("Apustu errepikatuaren komisioa ("+bezeroa+")",komisioa,
                       "irabazi");
7    }
8    return komisioa;
9  }
```

```
1  private void etendakoErrepikapena(Apustua bet, Bezeroa bezeroa) {
2    if (bet.getErrepikatua()!=null) {
3      Bezeroa erre=bet.getErrepikatua();
4      Errepikapena er=bezeroa.getErrepikapena(erre);
5      er.eguneratuHilHonetanGeratzenDena(bet.getKopurua());
6    }
7  }
```

Errefaktorizazio Prozedura

Kasu honetan, errefaktorizazioa egiteko eman behar izan diren pausuak errazak izan dira:

- Hasieran daukagun metodoa aztertu eta ikusi zein den/diren metodoak dituen aginduen artean multzokatu daitezkeen lerroak. Kasu honetan, multzo bakarra dago: erdialdean dauden for-ak.
- Metodo berri batean sartuko dugun kode zati guztia aukeratu eta Eclipse-k eskaintzen duen errefaktorizazio aukera erabili. Horretarako, *Refactor -> Extract Method…* aukeratuko dugu eta lehio berri batean zer itxurarekin geratuko den klasea ikusi ahal dugu.
- Guk egindako kasuan, metodoa privatu bezala adierazi dugu eta onartu eskaintzen duten aukera eta ikusi ahalko dugu nola automatikoki kodea aldatu egin den.

- Gero, prozesu berdina errepikatuko dugu, sortu ditugun metodo guztiak 15 lerro edo motzagoak izan daitezen.

Gainera, kontuan izan behar da ezabatuGertaera metodoarentzat proba klase bat sortu dugula, funtzionamenduaren zuzentasuna bermatu ahal izateko arazo barik. Horregatik, behin errefaktorizazioa amaitzerakoan, proba kasuak berriro exekutatu beharko ditugu ondo dagoela ziurtatzeko. Proba kasu hauek EzabatugGertaeraDAW.java klasean daude.

Keep Small Interfaces Unit

Errefaktorizazio honen helburua funtzio bati ematen zaizkion parametro kopuru maximoa 4 edo gutxiago izatea da. 4 parametrotik gorako funtzioetan egiten dena honakoa da: zenbait parametro klase berri bateko objektu batean gorde eta objektu hori pasatzen da parametro gisa, eta ondoren funtzioan zehar objektuko elementuak lortzen joango dira.

Errepikatu() funtzioan egingo dugu prozedura hau; izan ere, 5 parametro ematen baizaizkio DataAccess-eko funtzio honi. Beraz, lehendabizi funtzio honekin erlazionatuta dauden elementu guztiak erakutsiko ditugu eta errefaktorizazioa egin ostean geratu diren aldaketekin jarraituko dugu eta azkenik, egindako prozedura.

Aurretiko Kodeketa

Honako kodeketa ikusi daiteke errepikatu() funtzioarekin erlazio duen funtzio ororena:

BLFacadeInplementation klaseko kodea duzue jarraian,

```
@WebMethod
public void errepikatu(Bezeroa nork, Bezeroa nori, double apustatukoDena, double
hilabetekoMax, double komisioa) {
	dbManager.open(false);
	dbManager.errepikatu(nork, nori, apustatukoDena, hilabetekoMax, komisioa);
	dbManager.close();
}
```

Kasu honetan, DataAccess-ean nola inplementatuta dagoen ikusi daiteke,

```
public void errepikatu(Bezeroa nork, Bezeroa nori, double apustatukoDena, double
hilabetekoMax, double komisioa){
	Bezeroa errepikatzailea = db.find(Bezeroa.class, nork.getErabiltzaileIzena());
	Bezeroa errepikatua = db.find(Bezeroa.class, nori.getErabiltzaileIzena());
	db.getTransaction().begin();
	Errepikapena errepikapenBerria = errepikatua.addErrepikatzailea(nork,
apustatukoDena, hilabetekoMax, komisioa);
	errepikatzailea.addErrepikatua(errepikapenBerria);
	db.persist(errepikapenBerria);
	db.getTransaction().commit();
}
```

Bezeroa klaseko kodea ere aztertu beharra dago; izan ere, aldaketak jasango baititu errefaktorizazioa burutu ostean

```
public Errepikapena addErrepikatzailea(Bezeroa nork, double apustatukoDena, double
hilabetekoMax, double komisioa) {
	Errepikapena errepikapenBerria = new Errepikapena(nork, this, apustatukoDena,
	hilabetekoMax,komisioa);
	this.errepikatzaileak.add(errepikapenBerria);
```

```
        return errepikapenBerria;
}
```

Errefaktorizazioa burutu osteko kodea

Behin errefaktorizazio prozedurak egin ostean (hurrengo atalean azalduko da zer egin den), honako kodea gelditu zaigu aurretik erakutsitako klaseetan.

BLFacadeInplementation klasean bi lerro bat gehitu zaio, eta DataAccess klaseko funtzioaren deia ere aldatu dela ikusi daiteke,

```
@WebMethod
public void errepikatu(Bezeroa nork, Bezeroa nori, double apustatukoDena, double
hilabetekoMax, double komisioa) {
        dbManager.open(false);
        DatuErrefaktorizatuERREPIKAPEN a=new
DatuErrefaktorizatuERREPIKAPEN(apustatukoDena,hilabetekoMax,komisioa);
        dbManager.errepikatu(nork, nori, a);
        dbManager.close();
}
```

DataAccess klasean aldaketa handirik ez dago, metodoaren signaturan eta addErrepikatzailea() funtzioaren deian ikusten dira aldaketak,

```
public void errepikatu(Bezeroa nork, Bezeroa nori, DatuErrefaktorizatuERREPIKAPEN a){
        Bezeroa errepikatzailea = db.find(Bezeroa.class, nork.getErabiltzaileIzena());
        Bezeroa errepikatua = db.find(Bezeroa.class, nori.getErabiltzaileIzena());
        db.getTransaction().begin();
        Errepikapena errepikapenBerria = errepikatua.addErrepikatzailea(nork,a);
        errepikatzailea.addErrepikatua(errepikapenBerria);
        db.persist(errepikapenBerria);
        db.getTransaction().commit();
}
```

Bezeroa klasean ere, DataAccess klasean bezala, metodoaren signaturan aldatzen da, eta Errepikapena klaseko objektu berri bat sortzerakoan ere aldaketa aurkitzen da.

```
public Errepikapena addErrepikatzailea(Bezeroa nork, DatuErrefaktorizatuERREPIKAPEN a) {
        Errepikapena errepikapenBerria = new Errepikapena(nork, this,a);
        this.errepikatzaileak.add(errepikapenBerria);
        return errepikapenBerria;
}
```

Aurretik ikusitako aldaketak bi klase hauetan oinarritzen dira gehienbat: DatuErrefaktorizatuERREPIKAPEN klasearen sorrera eta Errepikapen objektuaren eraikitzailean. Jarraian erakutsiko dizuegu bakoitza nola kodetu dugun. DatuErrefaktorizatuERREPIKAPEN klaseak Errepikapen objektuak beharrezko dituen apustatukoDena, hilabetekoMax eta komisioa atributuak gordeko ditu, eta honako hau

izango litzateke bere kodeketa: oso sinplea da, atributuen erazagupena eta eraikitzailea nahikoak dira:

```
public class DatuErrefaktorizatuERREPIKAPEN {
	public double apustuko;
	public double hilabeteMax;
	public double komisio;

	public DatuErrefaktorizatuERREPIKAPEN(double a,double b,double c) {
		this.apustuko=a;
		this.hilabeteMax=b;
		this.komisio=c;
	}
}
```

Errepikapena klasearen kasuan, eraikitzaile berri bat sortuko dugu, hiru parametro dituena, eta azken hori sortu dugun klasearen objektua izango da; hau da, DatuErrefaktorizatuERREPIKAPEN klasekoa.

```
public Errepikapena(Bezeroa nork, Bezeroa nori, DatuErrefaktorizatuERREPIKAPEN a) {
	this.nork=nork;
	this.nori=nori;
	this.apustatukoDena=a.apustuko;
	this.hilabetekoMax=a.hilabeteMax;
	hilabeteHonetanGeratzenDena=a.hilabeteMax;
	this.komisioa=a.komisio;
}
```

Errefaktorizazio Prozedura

Kasu honetan jarraitu behar izan dugun prozedura honakoa izan da:

- ➢ DatuErrefaktorizatuERREPIKAPEN klasea sortu: horretarako atributuak erazagutu eta eraikitzailea definitu.
- ➢ Behin hori eginik, DataAccess-eko metodora joan gara eta bertan metodoaren signatura aldatu dugu: lehendik zeuden hiru atributu ezabatu() eta berria sartu. Horrela, BLFacadeInplementation-en dagoen metodoa aldatu dela ikusi daiteke.
- ➢ BLFacadeInplementation-en ordea, DatuErrefaktorizatuERREPIKAPEN klaseko objektu berria sortuko dugu.
- ➢ Errepikapena klasean eraikitzaile berria sortuko dugu hiru parametro dituena.
- ➢ Bezeroa klaseko addErrepikatzailea metodoaren signatura aldatuko dugu, jasotzen dituen parametroak ezabatuz eta DatuErrefaktorizatuERREPIKAPEN klaseko objektua hartzeko, eta Errepikatzaile berria sortzerako unean aurreko pausoan sortutako eraikitzailea erabiliko dugu.

Kontua da, zergatik sortzen duzu DatuErrefaktorizatuERREPIKAPEN klaseko objektua BLFacadeInplementation klasean? Bada, Internet-en informatzen ari naiz ea hiru/edozein mailako aplikazio batean DTO(Data Transfer Object) baten eraketa zein interfazetan egitea

den egokiena (azkenean, GUI batetik DataAccess-era joango dira errepikatu() funtzioak erabiltzen dituen datuak), eta Negozio Logikan gomendatzen dute; izan ere, zenbat eta beranduago sortu mota honetako objektuak hobea omen da kostu handikoak baitira horrelako egiturak. Beraz, BLFacadeInplementation-en egitea da egokiena, eta horrela egin dugu.

Laburbilduz, egin duguna izan da Refactor->Change Method Signature erabili eta horren bitartez DatuErrefaktorizatuERREPIKAPEN klaseko objektuak erabiltzeko esango diogu DataAccess-eko metodoei, eta ondoren datozen klaseei.

Proba kasuen inguruan, ErrepikatuDAW.java fitxategian ikusi daiteke proba kasua: ez dauka misterio handirik, proba kasu posible bakarra du eta ongi funtzionatzen duela ikusi daiteke.

OHARRAK:

Atal honetan aurki daitezke GitHub-eko eta SonarCloud-eko biltegien estekak, eta bertan aurkitu daitekeen informazioa:

- GitHub: https://github.com/Elola27/SI2_Integratua.git bertan aurki daiteke kodea eta aurretiko lanetako txostenak (iazko txosten lanak eta aurten sonar laborategia, proba kasuena eta azken hau)
- SonarCloud: https://sonarcloud.io/dashboard?id=SI2_Integratua Hemen aurkitu daiteke aztertu den kodea eta berari dagokion Analisia.