SOFTWARE INGENIARITZA II

PROIEKTUA

ERREFAKTORIZAZIOA

Eki Mendibil
Nikolas Vicuña
Beñat Sarasua

2021eko urriaren 17a

Eki Mendibil

DataAcces-eko “irabazi” metodoaren konplexutasun ziklomatikoa 4ra jeitsi dut. Hau egiteko, metodoaren zati bat metodo pribatu berri batean sartu dut. Zehazki, refactor ataleko “extract method” aukera baliatu dut lehenengo for begizta “ordaindu” metodo berrian sartzeko. Begizta horrek apostu zuzenak egin dituzten erabiltzaileei ordaintzeko erabiltzen da eta, ondorioz, baliteke programa gehiago garatu ezkero berriro erabili behar izatea. Metodo berria sortuaz, irabazi metodoaren konplexutasuna nabarmen murrizteaz gain, etorkizunean kode errepikatua gertatzea ekiditen laguntzen du.

Hau da metodoa aldaketak aplikatu baino lehen:

```
public void irabazi(Pronostico irabPronostico) {
    db.getTransaction().begin();
    Pronostico pronosticolocala= db.find(Pronostico.class, irabPronostico);
    RegisteredUser usuario;
    pronosticolocala.setEstado(true);//marco como resuelto el pronostico
    db.persist(pronosticolocala);
    for (Apostua i:pronosticolocala.getApostuak()) {
        if (i.pagar()){
            usuario=i.getUsuarioa();
            double cuota=1;
            for (Pronostico j:i.getPronostikoa()) {
                cuota*=j.getCuota();
            }
            usuario.setBalance(usuario.getBalance()+i.getKantitatea()*cuota);//Pago la cuota corespondiente
            if (i.getCopiado()!=null) {
                usuario.setBalance(usuario.getBalance()-i.getKantitatea()*cuota*i.getComision());
                i.getCopiado().setBalance(i.getCopiado().getBalance()+i.getKantitatea()*cuota*i.getComision());
                db.persist(i.getCopiado());
            }
            db.persist(usuario);
        }
    }//pago las apuestas que coresponde pagar
    for (Pronostico i: pronosticolocala.getGaldera().getPronostikoak()) {
        if (!i.getEstado()) {
            i.perder();
            db.persist(i);
            for (Apostua j:i.getApostuak()) {
                db.persist(j);
            }
        }
    }//el resto de pronosticos de la galdera los marco como perdidos
    db.getTransaction().commit();
}
```

Eta hau da metodoa aldaketak aplikatu ondoren “ordaindu” metodoarekin batera:

```
public void irabazi(Pronostico irabPronostico) {
    db.getTransaction().begin();
    Pronostico pronosticolocala= db.find(Pronostico.class, irabPronostico);
    RegisteredUser usuario;
    pronosticolocala.setEstado(true);//marco como resuelto el pronostico
    db.persist(pronosticolocala);
    ordaindu(pronosticolocala);//pago las apuestas que coresponde pagar
    for (Pronostico i: pronosticolocala.getGaldera().getPronostikoak()) {
        if (!i.getEstado()) {
            i.perder();
            db.persist(i);
            for (Apostua j:i.getApostuak()) {
                db.persist(j);
            }
        }
    }//el resto de pronosticos de la galdera los marco como perdidos
    db.getTransaction().commit();
}
```

```java
/**
 * @param pronosticolocala
 */
private void ordaindu(Pronostico pronosticolocala) {
    RegisteredUser usuario;
    for (Apostua i:pronosticolocala.getApostuak()) {
        if (i.pagar()){
            usuario=i.getUsuarioa();
            double cuota=1;
            for (Pronostico j:i.getPronostikoa()) {
                cuota*=j.getCuota();
            }
            usuario.setBalance(usuario.getBalance()+i.getKantitatea()*cuota);//Pago la cuota corespondiente
            if (i.getCopiado()!=null) {
                usuario.setBalance(usuario.getBalance()-i.getKantitatea()*cuota*i.getComision());
                i.getCopiado().setBalance(i.getCopiado().getBalance()+i.getKantitatea()*cuota*i.getComision());
                db.persist(i.getCopiado());
            }
            db.persist(usuario);
        }
    }
}
```

Metodo honi deia egiten dioten beste metodo eta klaseetan ez dira inolako aldaketarik egin, kode zati bat metodo berri batean sartzea errefaktorizazio bat delako eta, ondorioz, ez duelako metodoaren kanpo portaera aldatzen.

Beñat Sarasua

DataAccess-eko ApostuaEgin eta ApostuaEginDos metodoetako kode errepikatua murriztea. Bi metodo horiek oso antzekoak dira, beraz, ApostuaEgin klaseako kode errepikatua ezabatu eta ApostuaEginDos metodoari egingo diogu deia.

· ApostuaEgin metodoa:

```
public RegisteredUser ApostuaEgin(double DiruKantitaea,RegisteredUser usuario, Vector<Pronostico>  pronostico)
        throws IncorrectBetException{
    db.getTransaction().begin();
    Vector<Pronostico> pronosticolocal=new Vector<Pronostico>();
    Pronostico tmp;
    for (Pronostico i:pronostico) {
        tmp=db.find(Pronostico.class, i);
        if (tmp.getGaldera().getBetMinimum()>DiruKantitaea) {
            System.out.println("Froga bat da");
            throw new IncorrectBetException();
        }
        pronosticolocal.add(tmp);
    }
    RegisteredUser user = db.find(RegisteredUser.class, usuario.getUsername());
    Apostua apusta=new Apostua(DiruKantitaea, user, pronosticolocal);
    user.setBalance(usuario.getBalance()-DiruKantitaea);
    db.persist(user);
    db.persist(apusta);
    for (Pronostico i:pronosticolocal) {
        db.persist(i);
    }
    db.getTransaction().commit();
    for (RegisteredUser i:user.getJarraitzendidate()) {
        try {
            ApostuaEginDos(DiruKantitaea*i.getPortzentaia(user), i, pronosticolocal,user);
        }catch (IncorrectBetException e) {
        }
    }
    return user;
}
```

· ApostuaEginDos metodoa:

```
private RegisteredUser ApostuaEginDos (double DiruKantitaea,RegisteredUser usuario,
        Vector<Pronostico> pronostico,RegisteredUser mandon) throws IncorrectBetException{
    db.getTransaction().begin();
    Vector<Pronostico> pronosticolocal=new Vector<Pronostico>();
    Pronostico tmp;
    for (Pronostico i:pronostico) {
        tmp=db.find(Pronostico.class, i);
        if (tmp.getGaldera().getBetMinimum()>DiruKantitaea) {
            throw new IncorrectBetException();
        }
        pronosticolocal.add(tmp);
    }

    RegisteredUser user = db.find(RegisteredUser.class, usuario.getUsername());
    Apostua apusta=new Apostua(DiruKantitaea, user, pronosticolocal,mandon);
    user.setBalance(usuario.getBalance()-DiruKantitaea);
    db.persist(user);
    db.persist(apusta);
    for (Pronostico i:pronosticolocal) {
        db.persist(i);
    }
    db.getTransaction().commit();
    return user;
}
```

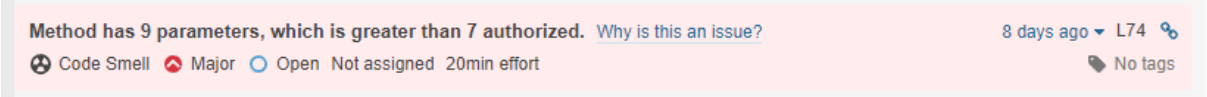
Aldaketak egin ondoren honela geratuko litzateke ApostuaEgin metodoa:

```
public RegisteredUser ApostuaEgin(double DiruKantitaea,RegisteredUser usuario, Vector<Pronostico>  pronostico)
    RegisteredUser user = this.ApostuaEginDos(DiruKantitaea, usuario, pronostico, null);
    for (RegisteredUser i:user.getJarraitzendidate()) {
        try {
            ApostuaEginDos(DiruKantitaea*i.getPortzentaia(user), i, pronostico,user);
        }catch (IncorrectBetException e) {
        }
    }
    return user;
}
```

ApostuaEginDos metodoan ez dugu aldaketarik egin behar izan. Test klaseek ere ez dute aldaketarik behar izan.

Nikolas Vicuña

Lan egingo dudan Code Smell-a, parametro kopurua 9tik, 4ra jaistea da DataAccesseko register metodoan.



Ikusi dezakegun bezala, **String Izena, String Id, String Email, String Pasahitza, int urtea, int hilabetea, int eguna, long BankuZenbakia, int tipo** parametroak dauzkagu. Parametro guzti horiek ekiditzeko, bi array sortuko ditugu, bat String motakoa eta bestea int motakoa eta beraietan hurrengo parametroak batuko ditugu. Horrela 4 parametro soilik erabiliko dira.

myStringList → Izena, Id, Email, Pasahitza
MyIntList → urtea, hilabetea, eguna

Ikusiko dugun bezala, DataAccess.java klasea aldatu ondoren, Interfazea, implementazio eta test motako klaseak ere aldatu egin behar direla. Testen kasuan, bakoitzeko adibide bat jarriko dugu eta ez klase osoa zeren oso luze geratuko litzateke, adibide batekin besteak ulertzen dira.

1.DataAccess.java klasean:
Aurretik azaldu dudan bezala, irudian ikus dezakegu 9 parametro ditugula. BankuZenbakia eta tipo parametroak soilik mantenduko ditugu eta beste guztiak parametro motako Array baten gordeko ditugu. Kodea ere pixkat moldatu beharko da, orain array-ekin egingo dugu lan eta.

Lehen bertsioa:

```
public void register(String Izena, String Id, String Email, String Pasahitza,
        int urtea, int hilabetea, int eguna, long BankuZenbakia,int tipo) throws UserAlreadyExist{//0=Admin,1=worker,2=register
    db.getTransaction().begin();
    try {
        if (tipo==2) {
            db.persist(new RegisteredUser(Izena, Pasahitza, Email, Id, BankuZenbakia, UtilDate.newDate(urtea, hilabetea, eguna))
        }
        if (tipo==1) {
            db.persist(new Worker(Izena, Pasahitza, Email, Id));
        }
        if (tipo==0) {
            db.persist(new Admin(Izena, Pasahitza, Email, Id));
        }
        db.getTransaction().commit();
        System.out.println("Saved");
    }
    catch(javax.persistence.RollbackException e){
        throw new UserAlreadyExist();
    }
}
```

Aldaketak egin ondoren honela geratzen da:

```
public void register(String[] myStringList, int[] myIntList, long BankuZenbakia,int tipo) throws UserAlreadyExist{//0=Admin,1=worker,2=reg
    db.getTransaction().begin();
    try {
        if (tipo==2) {
            db.persist(new RegisteredUser(myStringList[0], myStringList[1], myStringList[2], myStringList[3],
                    BankuZenbakia, UtilDate.newDate(myIntList[0], myIntList[1], myIntList[2])));
        }
        if (tipo==1) {
            db.persist(new Worker(myStringList[0], myStringList[1], myStringList[2], myStringList[3]));
        }
        if (tipo==0) {
            db.persist(new Admin(myStringList[0], myStringList[1], myStringList[2], myStringList[3]));
        }
        db.getTransaction().commit();
        System.out.println("Saved");
    }
    catch(javax.persistence.RollbackException e){
        throw new UserAlreadyExist();
    }
}
```

2. BlFacade interfazean:

Lehen bertsioa:

```
74      @WebMethod public void register(String Izena, String Id, String Email, String Pasahitza,
75              int urtea, int hilabetea, int eguna, long BankuZenbakia,int tipo) throws UserAlreadyExist;
```

Aldaketak egin ondoren honela geratzen da:

```
74      @WebMethod public void register(String[] myStringList,int[] myIntList, long BankuZenbakia,int tipo) throws UserAlreadyExist;
```

3. Blfacadeimplementation klasean:

Lehen bertsioa:

```
@WebMethod
public void register(String Izena, String Id, String Email, String Pasahitza,
        int urtea, int hilabetea, int eguna, long BankuZenbakia,int tipo) throws UserAlreadyExist {
    dbManager.open(false);
    dbManager.register(Izena, Id, Email, Pasahitza,  urtea,hilabetea,eguna,  BankuZenbakia, tipo);
    dbManager.close();
}
```

Aldaketak egin ondoren honela geratzen da:

```
@WebMethod
public void register(String[] myStringList,int[] myIntList, long BankuZenbakia,int tipo) throws UserAlreadyExist {
    dbManager.open(false);
    dbManager.register(myStringList, myIntList,  BankuZenbakia, tipo);
    dbManager.close();
}
```

4.Register.java fitxategia aldaketan egin orduko:

```
                if (BankuZenbakiaSartu.getText().matches("^[0-9]{10,12}$")) {

                    if (UserSelect.isSelected()) {
                        facade.register(IzenaSartu.getText(),IdSartu.getText(), EmailSartu.getText(), PasaitzaSartu.getText(),
                                ano, mes, dia , numbank,2);
                    }
                    if (AdminSelect.isSelected()) {
                        facade.register(IzenaSartu.getText(),IdSartu.getText(), EmailSartu.getText(), PasaitzaSartu.getText(),
                                ano, mes, dia , numbank,0);
                    }
                    if (WorkerSelect.isSelected()) {
                        facade.register(IzenaSartu.getText(),IdSartu.getText(), EmailSartu.getText(), PasaitzaSartu.getText(),
                                ano, mes, dia , numbank,1);
                    }
                    origen.setVisible(true);
                    frame.setVisible(false);
                }else {
```

Orain parametroak Array ezberdinetan sartu eta metodoa deitzeko orduan parametro gisa pasatuko dugu beraien kopurua murrizteko.

```
            if (BankuZenbakiaSartu.getText().matches("^[0-9]{10,12}$")) {

                String[] myStringArray = {IzenaSartu.getText(), IdSartu.getText(), EmailSartu.getText(),PasaitzaSartu.getText()};
                int[] MyIntArray = {ano,mes,dia};

                if (UserSelect.isSelected()) {
                    facade.register(myStringArray, MyIntArray , numbank,2);
                }
                if (AdminSelect.isSelected()) {
                    facade.register(myStringArray, MyIntArray , numbank,0);
                }
                if (WorkerSelect.isSelected()) {
                    facade.register(myStringArray, MyIntArray, numbank,1);
                }
                origen.setVisible(true);
                frame.setVisible(false);
            }else {
```

5. RegisterDAW.java klasean:

```java
    @Test
    //sut.irabazi:  The user is a worker.
    public void test2() {

            //define parameters
            String izena = "Nikolas";
            String id = "12345678L";
            String email = "ni@ni.com";
            String pasahitza = "123";
            int urtea = 1994;
            int hilabetea = 8;
            int eguna = 11;
            long bankuZenbakia = 1234567899;
            int tipo = 1;
            Worker user = new Worker(izena, pasahitza, email, id);
            //invoke System Under Test (sut)
                try {
                    sut.open(true);
                    sut.register(izena, id, email, pasahitza, urtea, hilabetea, eguna, bankuZenbakia, tipo);
                    sut.close();
                    assertTrue(true);

                }
                catch(Exception e) {
                    fail();
                }
                finally {
                    testDA.removeUser(user);
                }

        }
```

```java
    @Test
    //sut.irabazi:  The user is a worker.
    public void test2() {

            //define parameters
            String[] myStringList = {"Nikolas","12345678L","ni@ni.com","123"};
            int[] myIntList = {1994,8,11};
            long bankuZenbakia = 1234567899;
            int tipo = 1;
            Worker user = new Worker(myStringList[0], myStringList[1], myStringList[2], myStringList[3]);
            //invoke System Under Test (sut)
                try {
                    sut.open(true);
                    sut.register(myStringList, myIntList, bankuZenbakia, tipo);
                    sut.close();
                    assertTrue(true);

                }
                catch(Exception e) {
                    fail();
                }
                finally {
                    testDA.removeUser(user);
                }

        }
```

6. RegisterDAB.java klasean:

```java
    @Test
    //sut.irabazi:  The user is an admin.
    public void test3() {

            //define parameters
            String izena = "Nikolas";
            String id = "12345678L";
            String email = "ni@ni.com";
            String pasahitza = "123";
            int urtea = 1994;
            int hilabetea = 8;
            int eguna = 11;
            long bankuZenbakia = 1234567899;
            int tipo = 0;
            //invoke System Under Test (sut)
                try {
                    sut.open(true);
                    sut.register(izena, id, email, pasahitza, urtea, hilabetea, eguna, bankuZenbakia, tipo);;
                    sut.close();
                    assertTrue(true);

                }
                catch(Exception e) {
                    fail();
                }

           }
```

```java
    @Test
    //sut.irabazi:  The user is an admin.
    public void test3() {

            //define parameters
            String[] myStringList = {"Nikolas","12345678L","ni@ni.com","123"};
            int[] myIntList = {1994,8,11};
            long bankuZenbakia = 1234567899;
            int tipo = 0;
            //invoke System Under Test (sut)
                try {
                    sut.open(true);
                    sut.register(myStringList,myIntList, bankuZenbakia, tipo);;
                    sut.close();
                    assertTrue(true);

                }
                catch(Exception e) {
                    fail();
                }

           }
```

7. RegisterMockInt.java klasean:

```java
    @Test
    //sut.irabazi:  The user is a registeredUser.
    public void test1() {

            //define parameters
            String izena = "Nikolas";
            String id = "12345678L";
            String email = "ni@ni.com";
            String pasahitza = "123";
            int urtea = 1994;
            int hilabetea = 8;
            int eguna = 11;
            long bankuZenbakia = 1234567899;
            int tipo = 2;

            //invoke System Under Test (sut)
                try {
                    sut.register(izena, id, email, pasahitza, urtea, hilabetea, eguna, bankuZenbakia, tipo);
                    Mockito.verify(dataAccess, Mockito.times(1)).register(izena, id, email, pasahitza, urtea, hilabetea, eguna, bankuZenbakia,
                    assertTrue(true);

                }
                catch(Exception e) {
                    fail();
                }

          }
```

```java
    @Test
    //sut.irabazi:  The user is a registeredUser.
    public void test1() {

            //define parameters
            String[] myStringList = {"Nikolas","12345678L","ni@ni.com","123"};
            int[] myIntList = {1994,8,11};
            long bankuZenbakia = 1234567899;
            int tipo = 2;

            //invoke System Under Test (sut)
                try {
                    sut.register(myStringList,myIntList, bankuZenbakia, tipo);
                    Mockito.verify(dataAccess, Mockito.times(1)).register(myStringList,myIntList, bankuZenbakia, tipo);
                    assertTrue(true);

                }
                catch(Exception e) {
                    fail();
                }

          }
```