Refactorización IS2

2023/2024

Proyecto Bets

Autores:
Unai Artano
Asier Contreras
Martin Ian Horsfield

eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

Índice

Información previa

No ha sido fácil encontrar código a refactorizar que cumpliera las características solicitadas ya que la gran mayoría del código ya estaba optimizado en gran medida del curso anterior.
Por eso, no hemos podido llegar al mínimo solicitado en cada apartado ya que revisando el código entre los 3 no encontramos código que necesitase dicha refactorización a simple vista.

Enlaces

Github:
https://github.com/asiercontreras/IS2-RepBets

SonnarCloud:
https://sonarcloud.io/project/overview?id=proyectois2-bets_repbets

Reparto de horas

Nombre	Horas
Unai Artano	2
Asier Contreras	2
Martin Ian Horsfield	2

"Write short units of code" (capítulo 2)

Autor: Asier Contreras

createBet()

El método de createBet() tiene en total unas 30 líneas de código, lo que dificulta su mantenimiento y que un programador que no haya programado el método entienda lo que el propio método quiere hacer. Para ello, se ha dividido el método en métodos más pequeños que ninguno sobrepase las 15 líneas para facilitar su uso y mantenibilidad a futuro. La siguiente imagen muestra cómo era el método antes de ser refactorizado.

```
public float createBet(float bet, User usr, Forecast fr) throws NotEnoughMoneyException {
    System.out.println(">> DataAcces: CreateBet => quantity: " + bet + ", question: " + fr.getQuestion());

    User u = db.find(User.class, usr.getDni());
    Forecast f = db.find(Forecast.class, fr.getFrNum());
    Bet b = u.findBet(fr);
    float newValue;
    if (b != null)
        newValue = u.getWallet() + b.getBet() - bet;
    else
        newValue = u.getWallet() - bet;
    if (newValue < 0)
        throw new NotEnoughMoneyException("You do not have enough money");

    db.getTransaction().begin();
    if (b != null) {
        Bet dbBet = db.find(Bet.class, b.getId());
        u.setWallet(newValue);
        dbBet.setBet(bet);
    } else {
        b = new Bet(bet, u, f);
        u.setWallet(newValue);
        u.addBet(b);
        f.addBet(b);
        db.persist(b);
    }
    db.getTransaction().commit();
    return newValue;
}
```

Una vez el método ha sido refactorizado, obtenemos un método createBet() que no supera en este caso las 8 líneas de código. Para ello hemos creado dos métodos nuevos, getNewWalletValue(), que devolverá en casa de que el monedero tuviera dinero suficiente el valor que debería de quedar en la cartera una vez realizada la apuesta. Y también se ha tenido que crear el método realizarApuestaEnBD() que es el método que se encargará de ponerse en contacto con la BD y realizar los cambios necesarios para modificar la apuesta ya creada o añadir una nueva apuesta.
El nuevo createBet():

```
public float createBet(float bet, User usr, Forecast fr) throws NotEnoughMoneyException {
    System.out.println(">> DataAcces: CreateBet => quantity: " + bet + ", question: " + fr.getQuestion());
    User u = db.find(User.class, usr.getDni());
    Forecast f = db.find(Forecast.class, fr.getFrNum());
    Bet b = u.findBet(fr);
    float newValue = this.getNewWalletValue(u, b, bet);
    this.realizarApuestaEnBD(b, u, newValue, bet, f);
    return newValue;
}
```

getNewWalletValue():

```
private float getNewWalletValue(User u, Bet b, float bet) throws NotEnoughMoneyException {
    float newValue;
    if (b != null)
        newValue = u.getWallet() + b.getBet() - bet;
    else
        newValue = u.getWallet() - bet;
    if (newValue < 0)
        throw new NotEnoughMoneyException("You do not have enough money");
    return newValue;

}
```

realizarApuestaEnBD():

```
private void realizarApuestaEnBD(Bet b, User u, float newValue, float bet, Forecast f) {
    db.getTransaction().begin();
    if (b != null) {
        Bet dbBet = db.find(Bet.class, b.getId());
        u.setWallet(newValue);
        dbBet.setBet(bet);
    } else {
        b = new Bet(bet, u, f);
        u.setWallet(newValue);
        u.addBet(b);
        f.addBet(b);
        db.persist(b);
    }
    db.getTransaction().commit();
}
```

Una vez hecha la refactorización se ha realizado la clase de testing para verificar que todo funciona correctamente y que seguimos manteniendo una cobertura del 100%.

```
public float createBet(float bet, User usr, Forecast fr) throws NotEnoughMoneyException {
    System.out.println(">> DataAcces: CreateBet => quantity: " + bet + ", question: " + fr.getQuestion());
    User u = db.find(User.class, usr.getDni());
    Forecast f = db.find(Forecast.class, fr.getFrNum());
    Bet b = u.findBet(fr);
    float newValue = this.getNewWalletValue(u, b, bet);
    this.realizarApuestaEnBD(b, u, newValue, bet, f);
    return newValue;
}
```

```java
private float getNewWalletValue(User u, Bet b, float bet) throws NotEnoughMoneyException {
    float newValue;
    if (b != null)
        newValue = u.getWallet() + b.getBet() - bet;
    else
        newValue = u.getWallet() - bet;
    if (newValue < 0)
        throw new NotEnoughMoneyException("You do not have enough money");
    return newValue;

}

private void realizarApuestaEnBD(Bet b, User u, float newValue, float bet, Forecast f) {
    db.getTransaction().begin();
    if (b != null) {
        Bet dbBet = db.find(Bet.class, b.getId());
        u.setWallet(newValue);
        dbBet.setBet(bet);
    } else {
        b = new Bet(bet, u, f);
        u.setWallet(newValue);
        u.addBet(b);
        f.addBet(b);
        db.persist(b);
    }
    db.getTransaction().commit();
}
```

"Write simple units of code" (capítulo 3)

Revisando el capítulo 3 del libro recomendado por el profesorado, no hemos encontrado ningún apartado de código que se asemeje a lo pedido para refactorizar. Esto seguramente sea debido a que el proyecto se optimizó el año pasado a la hora de realizar el proyecto.

"Duplicate code" (capítulo 4)

Autor: Martín Horsfield

CreateForecastGUI.java

En la clase CreateForecastGUI.java llamamos al fichero Etiquetas varias veces para poder cambiar de idioma cuando queramos. La desventaja es que tenemos que hacer muchísimas llamadas al fichero. Esto significa que además de usar muchos recursos, si en algún momento queremos modificar el nombre del fichero o el funcionamiento, tendremos que cambiar muchas líneas de código.

```
23
24    public class CreateForecastGUI extends JFrame {
25            private static final long serialVersionUID = 1L;
26    +
27    +       private ResourceBundle RBEtiquetas = ResourceBundle.getBundle("Etiquetas");
28    +
29    +       private final JLabel jLabelEventDate = new JLabel(RBEtiquetas.getString("EventDate"));
30    +       private final JLabel jLabelQueries = new JLabel(RBEtiquetas.getString("Queries"));
```

Añadiendo este atributo, podemos realizar el refactor.
Pasaremos de esto:

```
32          private JLabel lblError;
33
34          private JButton jButtonClose = new JButton(ResourceBundle.getBundle("Etiquetas").getString("Close"));
35  -       private JButton btnCreateForecast = new JButton(ResourceBundle.getBundle("Etiquetas").getString("CreateForecast"));
36
37          // Code for JCalendar
```

A esto:

```
35
36  +       private JButton jButtonClose = new JButton(RBEtiquetas.getString("Close"));
37  +       private JButton btnCreateForecast = new JButton(RBEtiquetas.getString("CreateForecast"));
38
39          // Code for JCalendar
```

Este es el resto del código.

```
52  +       private String[] columnNamesEvents = new String[] { RBEtiquetas.getString("EventN"),
53  +                       RBEtiquetas.getString("Event"),
54
55          };
56          private String[] columnNamesForecast = new String[] {
57  +                       RBEtiquetas.getString("Description"),
58  +                       RBEtiquetas.getString("Winrate"), };
59
60          private JComboBox<Question> queryBox = new JComboBox<>();
61          private JTextField descriptionField;
62          private JTextField winrateField;
63  +       private final JLabel lblForecasts = new JLabel(RBEtiquetas.getString("Forecasts")); //$NON-NLS-1$ //$NON-NLS-2$
64          private JScrollPane scrollPaneForecast = new JScrollPane();
65          private JTable tableForecast = new JTable();
66
108                 this.getContentPane().setLayout(null);
109                 this.setSize(new Dimension(700, 500));
110                 this.setResizable(false);
111 +               this.setTitle(RBEtiquetas.getString("CreateForecast"));
112
113                 jLabelEventDate.setBounds(new Rectangle(40, 15, 140, 25));
114                 jLabelQueries.setBounds(40, 230, 350, 14);
192                                         Vector<domain.Event> events = facade.getEvents(firstDay);
193
```

```
194                                         if (events.isEmpty()) {
195 +                                               jLabelEvents.setText(RBEtiquetas.getString("NoEvents") + ": "
196                                                                             + dateformat1.format(calendarAct.getTime()));
197                                                 queryBox.removeAllItems();
198                                         } else
199 +                                               jLabelEvents.setText(RBEtiquetas.getString("Events") + ": "
200                                                                             + dateformat1.format(calendarAct.getTime()));
201                                         for (domain.Event ev : events) {
202                                                 Vector<Object> row = new Vector<Object>();
```

```
237
238                         if (queries.isEmpty())
239                                 jLabelQueries.setText(
240 +                                               RBEtiquetas.getString("NoQueries") + ": " + ev.getDescription());
241                         else
242 +                               jLabelQueries.setText(RBEtiquetas.getString("SelectedEvent") + " "
243                                                 + ev.getDescription());
244
245                         for (domain.Question q : queries) {
```

```
237
238                         if (queries.isEmpty())
239                                 jLabelQueries.setText(
240 +                                               RBEtiquetas.getString("NoQueries") + ": " + ev.getDescription());
241                         else
242 +                               jLabelQueries.setText(RBEtiquetas.getString("SelectedEvent") + " "
243                                                 + ev.getDescription());
244
245                         for (domain.Question q : queries) {
299                                 lblError.setForeground(new Color(255, 0, 0));
300                                 String description = descriptionField.getText();
301                                 if (description.length() <= 0) {
302 +                                       lblError.setText(RBEtiquetas.getString("DescriptionField"));
303                                         return;
304                                 }
305                                 Question q = (Question) queryBox.getSelectedItem();
315                                 winrateField.setText("");
316                                 descriptionField.setText("");
317                                 lblError.setForeground(new Color(0, 0, 0));
318 +                               lblError.setText(RBEtiquetas.getString("ForecastCreated"));
319                         } catch (NumberFormatException e1) {
320 +                               lblError.setText(RBEtiquetas.getString("WinrateField"));
321                         } catch (ObjectAlreadyExistException e1) {
322                                 lblError.setText(e1.getMessage());
323                         }
```

Con estos cambios, la mantenibilidad de nuestro código mejora considerablemente.

Autor: Asier Contreras

CreateBetGUI.java

En la clase CreateBetGUI.java tenemos también el mismo problema que en la clase anterior por lo que hemos realizado el mismo cambio. Hemos creado el siguiente atributo de la clase, para de esta manera, en caso de que en un futuro se cambie de nombre de fichero, sólo haya que hacerlo una vez.

```
private ResourceBundle RBEtiquetas = ResourceBundle.getBundle("Etiquetas");
```

De esta manera, hemos pasado el código de tenerlo así:

```
private final JLabel jLabelEventDate = new JLabel(ResourceBundle.getBundle("Etiquetas").getString("EventDate"));
private final JLabel jLabelQueries = new JLabel(ResourceBundle.getBundle("Etiquetas").getString("Queries"));
private final JLabel jLabelEvents = new JLabel(ResourceBundle.getBundle("Etiquetas").getString("Events"));
private final JLabel lblForecast = new JLabel(ResourceBundle.getBundle("Etiquetas").getString("Forecasts"));
private final JLabel labelBet = new JLabel(ResourceBundle.getBundle("Etiquetas").getString("Bet"));
```

A tenerlo como en la siguiente imagen. Hemos dejado de repetir el código ResourceBundle *getBundle* "Etiquetas" por la variable creada.

```
private final JLabel jLabelEventDate = new JLabel(RBEtiquetas.getString("EventDate"));
private final JLabel jLabelQueries = new JLabel(RBEtiquetas.getString("Queries"));
private final JLabel jLabelEvents = new JLabel(RBEtiquetas.getString("Events"));
private final JLabel lblForecast = new JLabel(RBEtiquetas.getString("Forecasts"));
private final JLabel labelBet = new JLabel(RBEtiquetas.getString("Bet"));
```

Autor: Unai Artano

WalletGUI.java

En la clase *WalletGUI.java* el código *new Font("Dialog", Font.BOLD, 16)* y *new Font("Dialog", Font.BOLD, 12)* se repite. Para ello, hemos creado dos variables; una para el tamaño 16 y la otra para el 12.
Antes de añadir las variables:

```
97                      JLabel lblError = new JLabel();
98                      lblError.setFont(new Font("Dialog", Font.BOLD, 12));
99                      lblError.setBounds(69, 175, 300, 20);
```

```
182             JLabel lblError = new JLabel();
183             lblError.setFont(new Font("Dialog", Font.BOLD, 12));
184             lblError.setBounds(174, 350, 300, 20);
```

```
238         JLabel lblSelectCard = new JLabel(ResourceBundle.getBundle("Etiquetas").getString("SelectSaved"));
239           lblSelectCard.setFont(new Font("Dialog", Font.BOLD, 16));
240           lblSelectCard.setBounds(85, 91, 340, 15);
```

```
266           JLabel lblNewLabel = new JLabel(ResourceBundle.getBundle("Etiquetas").getString("AddACard"));
267           lblNewLabel.setFont(new Font("Dialog", Font.BOLD, 16));
268           lblNewLabel.setBounds(84, 179, 278, 15);
```

Después de añadir las variables:

```
31      private Font font = new Font("Dialog", Font.BOLD, 16);
32      private Font font1 = new Font("Dialog", Font.BOLD, 12);
```

```
97          JLabel lblError = new JLabel();
98          lblError.setFont(font1);
99          lblError.setBounds(69, 175, 300, 20);
```

```
180          JLabel lblError = new JLabel();
181          lblError.setFont(font1);
182          lblError.setBounds(174, 350, 300, 20);
```

```
236       JLabel lblSelectCard = new JLabel(ResourceBundle.getBundle("Etiquetas").getString("SelectSaved"));
237         lblSelectCard.setFont(font);
238         lblSelectCard.setBounds(85, 91, 340, 15);
```

```
264         JLabel lblNewLabel = new JLabel(ResourceBundle.getBundle("Etiquetas").getString("AddACard"));
265         lblNewLabel.setFont(font);
266         lblNewLabel.setBounds(84, 179, 278, 15);
```

"Keep unit interfaces small" (capítulo 5)

Autor: Unai Artano

createUser()

El método de *createUser* se le pasa 6 parámetros: nombre, apellido, DNI, fecha de nacimiento, contraseña y si es administrador o no. No tiene mucho sentido tener tantos parámetros para que luego al llamar al método te devuelva el usuario que hayas creado. Por ello, se ha pensado que por parámetro se le pase un usuario anteriormente ya creado.
Por tanto, al aplicar este cambio en la clase *DataAccess.java*, también se han tenido que modificar otras clases (*BLFacade*, *BLFacadeImplementation* y *SingUPGUI*), sus respectivas clases de casos de prueba (*CreateBetDABTest* y *CreateBetDAWTest*) y la clase *ApplicationLauncher* a la hora de agregar los dos usuarios de ejemplo a la BD.

Clase *DataAccess.java*

Código inicial:

```
public User createUser(String name, String surnames, String dni, Date birthdate, String passwd, boolean isAdmin)
                throws ObjectAlreadyExistException {
        System.out.println(">> DataAccess: CreateUser=> name = " + name + " dni = " + dni);

        if (db.find(User.class, dni) != null)
                throw new ObjectAlreadyExistException("The user already exists");

        db.getTransaction().begin();
        User usr = new User(name, surnames, dni, birthdate, passwd, isAdmin);
        db.persist(usr);
        db.getTransaction().commit();
        return usr;
}
```

Código después de aplicar refactorización:

```
public User createUser(User u)
        throws ObjectAlreadyExistException {
    System.out.println(">> DataAccess: CreateUser=> name = " + u.getName() + " dni = " + u.getDni());

    if (db.find(User.class, u.getDni()) != null)
        throw new ObjectAlreadyExistException("The user already exists");

    db.getTransaction().begin();
    db.persist(u);
    db.getTransaction().commit();
    return u;
}
```

Clase *BLFacade.java*

Código inicial:

```
@WebMethod public User createUser(String name, String surnames, String dni, Date birthdate, char[] passwd, boolean isAdmin) throws ObjectAlreadyExistException, NoSuchAlgorithmException;
```

Código después de aplicar refactorización:

```
@WebMethod public User createUser(User u) throws ObjectAlreadyExistException, NoSuchAlgorithmException;
```

Clase *BLFacadeImplementation.java*

Código inicial:

```
@WebMethod
public User createUser(String name, String surnames, String dni, Date birthdate, char[] passwd, boolean isAdmin)
                throws ObjectAlreadyExistException, NoSuchAlgorithmException {
        User usr = null;
        String pass = String.valueOf(passwd);

        String hash = hashPass(pass);
        System.out.println("hashed passwd: " + hash);
        pass = null;
        passwd = null;

        dbManager.open(false);

        try {
                usr = dbManager.createUser(name, surnames, dni, birthdate, hash, isAdmin);
        } catch (ObjectAlreadyExistException e) {
                throw e;
        } finally {
                dbManager.close();
        }

        return usr;
}
```

Código después de aplicar refactorización:

```
@WebMethod
public User createUser(User u)
        throws ObjectAlreadyExistException, NoSuchAlgorithmException {
    User usr = null;
    String pass = String.valueOf(u.getPasswd());

    String hash = hashPass(pass);
    System.out.println("hashed passwd: " + hash);
    pass = null;
    u.setPasswd(null);

    dbManager.open(false);

    try {
        usr = dbManager.createUser(new User (u.getName(), u.getSurnames(), u.getDni(), u.getBirthdate(), hash, u.isAdmin()));
    } catch (ObjectAlreadyExistException e) {
        throw e;
    } finally {
        dbManager.close();
    }

    return usr;
}
```

Clase *SingUpGUI.java*

Código inicial:

```
if(name.length() > 0 && sur1.length() > 0 && sur2.length() > 0 &&
        dni.length() > 0 && passwd.length > 0){
    try {
        facade.createUser(name, surnames, dni, birthdate, passwd, false);
        jButtonClose_actionPerformed(e);
    } catch (ObjectAlreadyExistException e1) {
        lblerror.setText(e1.getMessage());
    } catch (NoSuchAlgorithmException e1) {
        lblerror.setText(e1.getMessage());
    }
}else lblerror.setText(ResourceBundle.getBundle("Etiquetas").getString("Empty"));
```

Código después de aplicar refactorización:

```
if(name.length() > 0 && sur1.length() > 0 && sur2.length() > 0 &&
        dni.length() > 0 && passwd.length > 0){
    try {
        String passS="";
        for(char c:passwd) {
            passS+=c;
        }
        facade.createUser(new User (name, surnames, dni, birthdate, passS, false));
        jButtonClose_actionPerformed(e);
    } catch (ObjectAlreadyExistException e1) {
        lblerror.setText(e1.getMessage());
    } catch (NoSuchAlgorithmException e1) {
        lblerror.setText(e1.getMessage());
    }
}else lblerror.setText(ResourceBundle.getBundle("Etiquetas").getString("Empty"));
```

Se ha creado un *forEach* porque si le pasabas esa variable como parámetro da error porque te pide un *String* y no un *char[]*. Al recorrer el vector y añadiendo cada elemento creas un *String* que luego lo pasarás como parámetro.

Clase *ApplicationLauncher.java*

Código inicial:

```
DataAccess da = new DataAccess(c.getDataBaseOpenMode().equals("initialize"));
appFacadeInterface = new BLFacadeImplementation(da);
char[] passwd = { '1', '1' };
appFacadeInterface.createUser("asier", "contreras", "11", new Date(2000, 5, 1), passwd, false);
appFacadeInterface.createUser("Adminitrador", "Administrador", "00", new Date(2000, 5, 1), passwd, true);
```

Código después de aplicar refactorización:

```
DataAccess da = new DataAccess(c.getDataBaseOpenMode().equals("initialize"));
appFacadeInterface = new BLFacadeImplementation(da);
char[] passwd = { '1', '1' };
String passS = "";
for(char ch: passwd) {
    passS+=ch;
}
appFacadeInterface.createUser(new User ("asier", "contreras", "11", new Date(2000, 5, 1), passS, false));
appFacadeInterface.createUser(new User ("Adminitrador", "Administrador", "00", new Date(2000, 5, 1), passS, true));
```

Se ha creado un *forEach* porque si le pasabas esa variable como parámetro da error porque te pide un *String* y no un *char[]*. Al recorrer el vector y añadiendo cada elemento creas un *String* que luego lo pasarás como parámetro.

Clases *CreateBetDABTest.java* y *CreateBetDAWTest.java*

Código inicial:

```
User user1 = sut.createUser("asier", "contreras", "24", new Date(2000, 5, 1), "11", false);
```

Código después de aplicar refactorización:

```
User user1 = sut.createUser(new User ("asier", "contreras", "24", new Date(2000, 5, 1), "11", false));
```

NOTA: Para todos los casos de pruebas que haya en las dos clases, la variable *user1* será la misma. Por tanto, sólo se mostrará un cambio aquí en el documento, pero en el código se han aplicado para todos los restantes.