Proyecto Bets22

Refactorización

UPV-EHU Ingeniería del Software II

Anal Lucía Durán Lengo

Jon Ortega Goikoetxea

Ana Lucía Durán

"Write short units of code"

- **Código original:**

El método `EmaitzakIpini()` ejecuta muchas acciones, siendo una de ellas el de actualizar los estados de todas las apuestas de una cuota. Como es una función diferenciable, se pueden extraer esos pasos para crear un método que se encargue de ello.

```
public void EmaitzakIpini(Quote quote) throws EventNotFinished{

    Quote q = db.find(Quote.class, quote);
    String result = q.getForecast();

    if(new Date().compareTo(q.getQuestion().getEvent().getEventDate())<0)
        throw new EventNotFinished();

    Vector<Apustua> listApustuak = q.getApustuak();
    db.getTransaction().begin();
    Question que = q.getQuestion();
    Question question = db.find(Question.class, que);
    question.setResult(result);
    for(Quote quo: question.getQuotes()) {
        for(Apustua apu: quo.getApustuak()) {

            Boolean b=apu.galdutaMarkatu(quo);
            if(b) {
                apu.getApustuAnitza().setEgoera("galduta");
            }else {
                apu.setEgoera("irabazita");
            }
        }
    }
    db.getTransaction().commit();
    for(Apustua a : listApustuak) {
        db.getTransaction().begin();
        Boolean bool=a.getApustuAnitza().irabazitaMarkatu();
        db.getTransaction().commit();
        if(bool) {
            this.ApustuaIrabazi(a.getApustuAnitza());
        }
    }
}
```

- **Código refactorizado:**

En su lugar, se hace la llamada al nuevo método `updateApustuak()` que actualiza todas las apuestas.

```java
public void EmaitzakIpini(Quote quote) throws EventNotFinished{

    Quote q = db.find(Quote.class, quote);
    String result = q.getForecast();

    if(new Date().compareTo(q.getQuestion().getEvent().getEventDate())<0)
        throw new EventNotFinished();

    Vector<Apustua> listApustuak = q.getApustuak();
    db.getTransaction().begin();
    Question que = q.getQuestion();
    Question question = db.find(Question.class, que);
    updateApustuak(result, question);
    db.getTransaction().commit();
    for(Apustua a : listApustuak) {
        db.getTransaction().begin();
        Boolean bool=a.getApustuAnitza().irabazitaMarkatu();
        db.getTransaction().commit();
        if(bool) {
            this.ApustuaIrabazi(a.getApustuAnitza());
        }
    }
}
```

```java
private void updateApustuak(String result, Question question) {
    question.setResult(result);
    for(Quote quo: question.getQuotes()) {
        for(Apustua apu: quo.getApustuak()) {

            Boolean b=apu.galdutaMarkatu(quo);
            if(b) {
                apu.getApustuAnitza().setEgoera("galduta");
            }else {
                apu.setEgoera("irabazita");
            }
        }
    }
}
```

"Write simple units of code"

- **Código original:**

El método EmaitzakIpini() anterior tenía una complejidad ciclomática de 6 (es decir, 5 *"branches"* posibles).

```
public void EmaitzakIpini(Quote quote) throws EventNotFinished{

    Quote q = db.find(Quote.class, quote);
    String result = q.getForecast();

    if(new Date().compareTo(q.getQuestion().getEvent().getEventDate())<0)
        throw new EventNotFinished();

    Vector<Apustua> listApustuak = q.getApustuak();
    db.getTransaction().begin();
    Question que = q.getQuestion();
    Question question = db.find(Question.class, que);
    question.setResult(result);
    for(Quote quo: question.getQuotes()) {
        for(Apustua apu: quo.getApustuak()) {

            Boolean b=apu.galdutaMarkatu(quo);
            if(b) {
                apu.getApustuAnitza().setEgoera("galduta");
            }else {
                apu.setEgoera("irabazita");
            }
        }
    }
    db.getTransaction().commit();
    for(Apustua a : listApustuak) {
        db.getTransaction().begin();
        Boolean bool=a.getApustuAnitza().irabazitaMarkatu();
        db.getTransaction().commit();
        if(bool) {
            this.ApustuaIrabazi(a.getApustuAnitza());
        }
    }
}
```

- **Código refactorizado:**

Tras hacer la extracción del método `updateApustuak()` en el punto anterior, también se consigue reducir la complejidad ciclomática a 4 (3 *"branches"*), por lo que se simplifica mucho la complejidad.

```
public void EmaitzakIpini(Quote quote) throws EventNotFinished{

    Quote q = db.find(Quote.class, quote);
    String result = q.getForecast();

    if(new Date().compareTo(q.getQuestion().getEvent().getEventDate())<0)
        throw new EventNotFinished();

    Vector<Apustua> listApustuak = q.getApustuak();
    db.getTransaction().begin();
    Question que = q.getQuestion();
    Question question = db.find(Question.class, que);
    updateApustuak(result, question);
    db.getTransaction().commit();
    for(Apustua a : listApustuak) {
        db.getTransaction().begin();
        Boolean bool=a.getApustuAnitza().irabazitaMarkatu();
        db.getTransaction().commit();
        if(bool) {
            this.ApustuaIrabazi(a.getApustuAnitza());
        }
    }
}
```

"Duplicate code"

- **Código original:**

Como se puede observar en la imagen siguiente, teníamos el Bad Smell de código duplicado, ya que utilizábamos en repetidas ocasiones, en este caso, "DiruaSartu", en lugar de utilizar, por ejemplo, una misma variable para poner repetidas veces el mismo texto.

```
        db.getTransaction().commit();

        this.DiruaSartu(reg1, 50.0, new Date(), "DiruaSartu");
        this.DiruaSartu(reg2, 50.0, new Date(), "DiruaSartu");
        this.DiruaSartu(reg3, 50.0, new Date(), "DiruaSartu");
        this.DiruaSartu(reg4, 50.0, new Date(), "DiruaSartu");

        System.out.println("Db initialized");
    }
    catch (Exception e){
        e.printStackTrace();
    }
}
```

- **Código refactorizado:**

Como hemos mencionado anteriormente, para solucionar el Bad Smell anterior, hemos creado una variable llamada “meter_dinero”, para así evitar la repetición de código. Puede verse en la siguiente imagen:

```
            String meter_dinero= "DiruaSartu";

            this.DiruaSartu(reg1, 50.0, new Date(), meter_dinero);
            this.DiruaSartu(reg2, 50.0, new Date(), meter_dinero);
            this.DiruaSartu(reg3, 50.0, new Date(), meter_dinero);
            this.DiruaSartu(reg4, 50.0, new Date(), meter_dinero);

            System.out.println("Db initialized");
        }
        catch (Exception e){
            e.printStackTrace();
        }
    }
```

“Keep unit interfaces small”

- **Código original:**

En este método, tenemos 5 parámetros de entrada, por lo que debemos hacer que este número sea 4 o menos, para poder corregir el Bad Smell.

```
public boolean mezuaBidali(User igorlea, String hartzailea, String titulo, String test, Elkarrizketa elkarrizketa)
    User igorle = db.find(User.class, igorlea.getUsername());
    User hartzaile = db.find(User.class, hartzailea);
    Elkarrizketa elk=null;
    if(hartzaile==null) {
        return false;
    }else {
        db.getTransaction().begin();
        Message m = new Message(igorle, hartzaile, test);
        db.persist(m);
        if(elkarrizketa!=null) {
            elk = db.find(Elkarrizketa.class, elkarrizketa.getElkarrizketaNumber());
        }else {
            elk= new Elkarrizketa(titulo, igorle, hartzaile);
            db.persist(elk);
            m.setElkarrizketa(elk);
            igorle.addElkarrizketak(elk);
            hartzaile.addElkarrizketak(elk);
        }
        elk.addMezua(m);
        igorle.addBidalitakoMezuak(m);
        hartzaile.addJasotakoMezuak(m);
        db.getTransaction().commit();
        return true;
    }
}
```

- **Código refactorizado:**

Para solucionar el problema visto anteriormente, he reducido los parámetros a 3 utilizando la clase MezuakContainer, la cual posee los 3 parámetros igorlea, hartzailea y elkarrizketa, por lo que simplemente hay que sustituirlos por la clase mencionada anteriormente, y el Bad Smell estará corregido.

```
public boolean mezuaBidali(MezuakContainer mensaje, String titulo, String test) {
    User igorle = db.find(User.class, mensaje.getIgorlea().getUsername());
    User hartzaile = db.find(User.class, mensaje.getHartzailea());
    Elkarrizketa elk=null;
    if(hartzaile==null) {
        return false;
    }else {
        db.getTransaction().begin();
        Message m = new Message(igorle, hartzaile, test);
        db.persist(m);
        if(mensaje.getElkarrizketa()!=null) {
            elk = db.find(Elkarrizketa.class, mensaje.getElkarrizketa().getElkarrizketaNumber());
        }else {
            elk= new Elkarrizketa(titulo, igorle, hartzaile);
            db.persist(elk);
            m.setElkarrizketa(elk);
            igorle.addElkarrizketak(elk);
            hartzaile.addElkarrizketak(elk);
        }
        elk.addMezua(m);
        igorle.addBidalitakoMezuak(m);
        hartzaile.addJasotakoMezuak(m);
        db.getTransaction().commit();
        return true;
    }
}
```

Jon Ortega

"Write short units of code"

- **Código original:**

El método `gertaerakSortu()` contiene un bloque de código que ejecuta una función diferenciable. En concreto, son los pasos que hay que seguir para rellenar y crear un nuevo objeto Event.

```
public boolean gertaerakSortu(String description,Date eventDate, String sport) {
    boolean b = true;
    db.getTransaction().begin();
    Sport spo =db.find(Sport.class, sport);
    if(spo!=null) {
        TypedQuery<Event> Equery = db.createQuery("SELECT e FROM Event e WHERE e.getEventDate() =?1 ",Event.class);
        Equery.setParameter(1, eventDate);
        for(Event ev: Equery.getResultList()) {
            if(ev.getDescription().equals(description)) {
                b = false;
            }
        }
        if(b) {
            String[] taldeak = description.split("-");
            Team lokala = new Team(taldeak[0]);
            Team kanpokoa = new Team(taldeak[1]);
            Event e = new Event(description, eventDate, lokala, kanpokoa);
            e.setSport(spo);
            spo.addEvent(e);
            db.persist(e);
        }
    }else {
        return false;
    }
    db.getTransaction().commit();
    return b;
}
```

- **Código refactorizado:**

Creando un nuevo método `fillEvent()`, se simplifica el código original y se facilita su lectura. El nuevo método se encarga de crear y devolver una nueva instancia de Event.

```
public boolean gertaerakSortu(String description,Date eventDate, String sport) {
    boolean b = true;
    db.getTransaction().begin();
    Sport spo =db.find(Sport.class, sport);
    if(spo!=null) {
        TypedQuery<Event> Equery = db.createQuery("SELECT e FROM Event e WHERE e.getEventDate() =?1 ",Event.class);
        Equery.setParameter(1, eventDate);
        for(Event ev: Equery.getResultList()) {
            if(ev.getDescription().equals(description)) {
                b = false;
            }
        }
        if(b) {
            Event e = fillEvent(description, eventDate, spo);
            db.persist(e);
        }
    }else {
        return false;
    }
    db.getTransaction().commit();
    return b;
}
```

```
private Event fillEvent(String description, Date eventDate, Sport spo) {
    String[] taldeak = description.split("-");
    Team lokala = new Team(taldeak[0]);
    Team kanpokoa = new Team(taldeak[1]);
    Event e = new Event(description, eventDate, lokala, kanpokoa);
    e.setSport(spo);
    spo.addEvent(e);
    return e;
}
```

"Write simple units of code"

- **Código original:**

El método gertaeraEzabatu() tiene 8 *"branches"* posibles, por lo que hace que el código sea complejo y difícil de leer. Se pueden extraer bloques de código que ejecuten acciones diferenciables para poder crear nuevos métodos, reduciendo así la complejidad.

```
public boolean gertaeraEzabatu(Event ev) {
    Event event  = db.find(Event.class, ev);
    boolean resultB = true;
    List<Question> listQ = event.getQuestions();

    for(Question q : listQ) {
        if(q.getResult() == null) {
            resultB = false;
        }
    }
    if(resultB == false) {
        return false;
    }else if(new Date().compareTo(event.getEventDate())<0) {
        TypedQuery<Quote> Qquery = db.createQuery("SELECT q FROM Quote q WHERE q.getQuestion().getEvent
        Qquery.setParameter(1, event.getEventNumber());
        List<Quote> listQUO = Qquery.getResultList();
        for(int j=0; j<listQUO.size(); j++) {
            Quote quo = db.find(Quote.class, listQUO.get(j));
            for(int i=0; i<quo.getApustuak().size(); i++) {
                ApustuAnitza apustuAnitza = quo.getApustuak().get(i).getApustuAnitza();
                ApustuAnitza ap1 = db.find(ApustuAnitza.class, apustuAnitza.getApustuAnitzaNumber());
                db.getTransaction().begin();
                ap1.removeApustua(quo.getApustuak().get(i));
                db.getTransaction().commit();
                if(ap1.getApustuak().isEmpty() && !ap1.getEgoera().equals("galduta")) {
                    this.apustuaEzabatu(ap1.getUser(), ap1);
                }else if(!ap1.getApustuak().isEmpty() && ap1.irabazitaMarkatu()){
                    this.ApustuaIrabazi(ap1);
                }
                db.getTransaction().begin();
                Sport spo =quo.getQuestion().getEvent().getSport();
                spo.setApustuKantitatea(spo.getApustuKantitatea()-1);
                db.getTransaction().commit();
            }
        }

    }
    db.getTransaction().begin();
    db.remove(event);
    db.getTransaction().commit();
    return true;
}
```

- **Código refactorizado:**

En este caso, se han creado dos métodos nuevos: `checkResults()`, que comprueba si algún resultado del evento es null, y `purgeApustuak()`, que se encarga de repasar las apuestas y eliminar aquellas que estén vacías o tengan el estado como “galduta”.

```
public boolean gertaeraEzabatu(Event ev) {
    Event event  = db.find(Event.class, ev);
    boolean resultB = checkResults(event);
    if(resultB == false) {
        return false;
    }else if(new Date().compareTo(event.getEventDate())<0) {

        TypedQuery<Quote> Qquery = db.createQuery("SELECT q FROM Quote q WHERE q.getQuestion().getEven
        Qquery.setParameter(1, event.getEventNumber());
        List<Quote> listQUO = Qquery.getResultList();
        for(int j=0; j<listQUO.size(); j++) {
            Quote quo = db.find(Quote.class, listQUO.get(j));
            for(int i=0; i<quo.getApustuak().size(); i++) {
                ApustuAnitza apustuAnitza = quo.getApustuak().get(i).getApustuAnitza();
                ApustuAnitza ap1 = db.find(ApustuAnitza.class, apustuAnitza.getApustuAnitzaNumber());
                db.getTransaction().begin();
                ap1.removeApustua(quo.getApustuak().get(i));
                db.getTransaction().commit();
                purgeApustuak(ap1);
                db.getTransaction().begin();
                Sport spo =quo.getQuestion().getEvent().getSport();
                spo.setApustuKantitatea(spo.getApustuKantitatea()-1);
                db.getTransaction().commit();
            }
        }

    }
    db.getTransaction().begin();
    db.remove(event);
    db.getTransaction().commit();
    return true;
}
```

```
private boolean checkResults(Event event) {
    boolean resultB = true;
    List<Question> listQ = event.getQuestions();

    for(Question q : listQ) {
        if(q.getResult() == null) {
            resultB = false;
        }
    }
    return resultB;
}
```

```
private void purgeApustuak(ApustuAnitza ap1) {
    if(ap1.getApustuak().isEmpty() && !ap1.getEgoera().equals("galduta")) {
        this.apustuaEzabatu(ap1.getUser(), ap1);
    }else if(!ap1.getApustuak().isEmpty() && ap1.irabazitaMarkatu()){
        this.ApustuaIrabazi(ap1);
    }
}
```

"Duplicate code"

- **Código original:**

En este caso ocurre lo mismo que en el de mi compañera, hay un String repetido en 3 ocasiones ("¿Quién ganará el partido?"), por lo que se crea un Bad Smell.

```
if (Locale.getDefault().equals(new Locale("es"))) {
    q1=ev1.addQuestion("¿Quién ganará el partido?",1);
    q2=ev1.addQuestion("¿Quién meterá el primer gol?",2);
    q3=ev11.addQuestion("¿Quién ganará el partido?",1);
    q4=ev11.addQuestion("¿Cuántos goles se marcarán?",2);
    q5=ev17.addQuestion("¿Quién ganará el partido?",1);
    q6=ev17.addQuestion("¿Habrá goles en la primera parte?",2);
```

- **Código refactorizado:**

Al crear la variable "ganar_partido" y sustituirla por el String repetido nombrado anteriormente, el Bad Smell se corregirá.

```
String ganar_partido = "¿Quién ganará el partido?";

if (Locale.getDefault().equals(new Locale("es"))) {
    q1=ev1.addQuestion(ganar_partido,1);
    q2=ev1.addQuestion("¿Quién meterá el primer gol?",2);
    q3=ev11.addQuestion(ganar_partido,1);
    q4=ev11.addQuestion("¿Cuántos goles se marcarán?",2);
    q5=ev17.addQuestion(ganar_partido,1);
    q6=ev17.addQuestion("¿Habrá goles en la primera parte?",2);

}
```

"Keep unit interfaces small"

- **Código original:**

En este método, tenemos 3 parámetros de entrada y queremos reducir este número. No es necesariamente un Bad Smell, pero no encontramos más métodos que contuvieran más de 4 parámetros.

```
public void storeRegistered(String username, String password, Integer bankAccount) {
    db.getTransaction().begin();
    Registered ad = new Registered(username, password, bankAccount);
    db.persist(ad);
    db.getTransaction().commit();
}
```

- **Código refactorizado:**

Para reducir el número de parámetros de entrada, he utilizado la clase User, la cual contenía los 3 parámetros de entrada del método storeRegistered.

```
public void storeRegistered(User registrado) {
    db.getTransaction().begin();
    Registered ad = new Registered(registrado.getUsername(), registrado.getPassword(), registrado.getBankAccount());
    db.persist(ad);
    db.getTransaction().commit();
}
```