Pruebas Software
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:

Equipo:

El equipo para este proyecto ha sido desarrollado por Unai Artano, Asier Contreras, y Martin Ian Horsfield durante el curso académico 2023/2024 para la asignatura Ingeniería del Software II.

Reparto de horas

Nombre	Horas
Unai Artano	8
Asier Contreras	8
Martin Ian Horsfield	8

Enlaces

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

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

Métodos Analizados:

createBet(): Asier Contreras

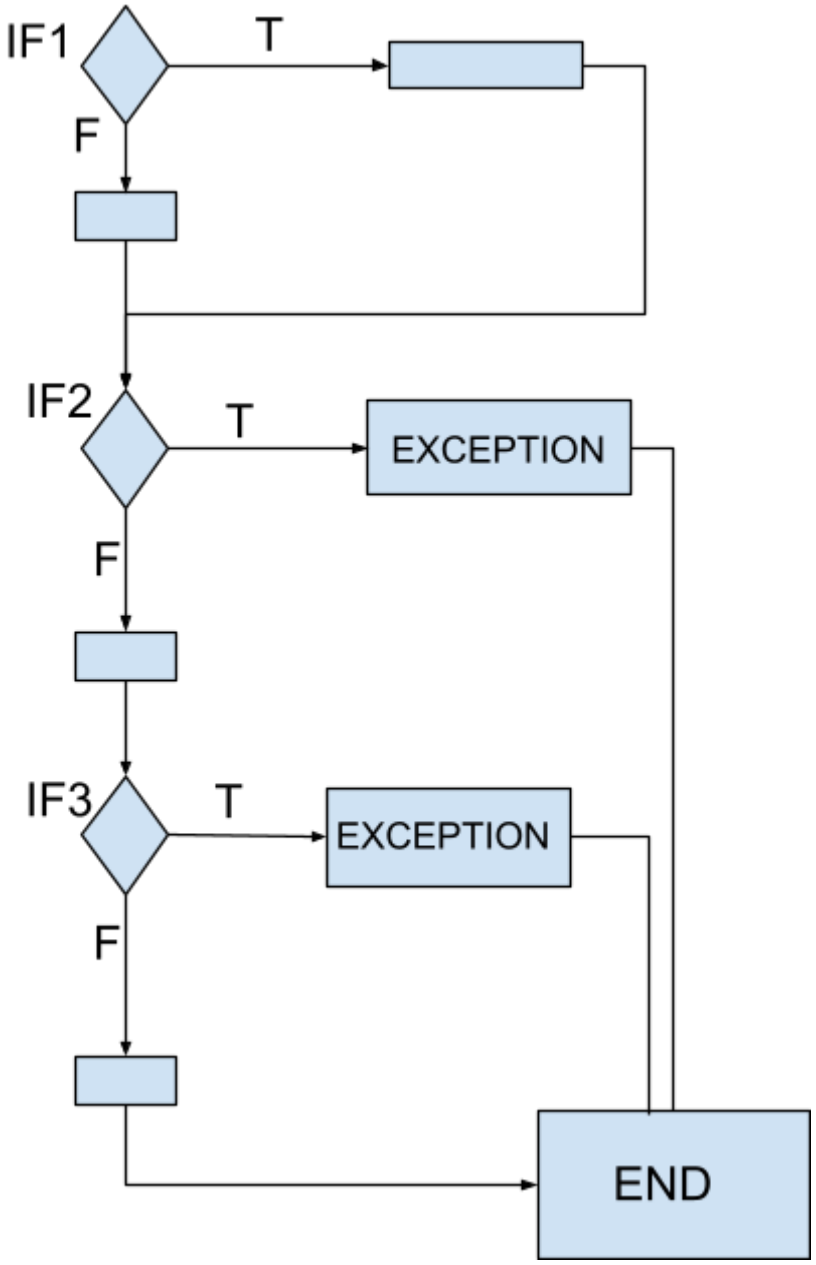
Código:

```
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;
}
```

Pruebas Unitarias:

Caja Blanca:

Grafo



Complejidad ciclomática

V(G)= #vértices- #nodos+ 2

$$V(G) = 12 - 10 + 2 = 4$$

V(G)= número de áreas cerradas+ 1

$$V(G) = 3 + 1 = 4$$

V(G)= #nodos condición+ 1

$$V(G) = 3 + 1 = 4$$

Caminos

Camino #1: IF1(T), IF2(T), END
Camino #2: IF1(F), IF2(T),END
Camino #3: IF1(T), IF2(F), IF3(T), END
Camino #4: IF1(T), IF2(F), IF3(F), END

Casos de prueba

#	Camino	Cond. Entrada	Entrada		Resultado Esperado	
			Parámetros	Estado BD	Salida	Estado BD
1	IF1(T), IF2(T), END	bet∈BD y no tiene dinero suficiente	(7,usr,"Gana Atleti")	u∈BD	Exception	Sin Cambio
2	IF1(F), IF2(T), END	bet∉BD y no tiene dinero suficiente	(7,usr,"Gana Atleti")	u∈BD	Exception	Sin Cambio
3	IF1(T), IF2(F), IF3(T), END	bet∈BD y tiene dinero suficiente	(5,usr,"Gana Atleti")	u∈BD		Modifica la apuesta que había
4	IF1(T), IF2(F), IF3(F), END	bet∉BD y tiene dinero suficiente	(5,usr,"Gan a Atleti")	u∈BD		Crea una nueva apuesta

```
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 realizada la ejecución de JUnit test con sus respectivos tests, obtenemos una cobertura del 100% para el método de createBet().

Caja Negra:

Clases de equivalencia

Condición entrada	Clases de equivalencia válida	Clases de equivalencia no válida
Saldo suficiente	(monedero- apuesta) >=0 (1)	(monedero- apuesta) <0 (2)
El usuario está en la bd	user ∈ BD (3)	user ∉ BD (4)
Realizar apuesta con apuesta previa	findBet()!= null (5)	
Realizar puesta sin apuesta previa	findBet()== null (6)	

Batería de casos de prueba

#	Clases de equivalencia cubiertas	Entrada		Resultado Esperado	
		Parámetros	Estado BD	Salida	Estado DB
1	2,3,6	(50,usr,"Gana Atleti")	usr∈BD	Error	Sin Cambios
2	1,4	(50,usr2,"Gana Atleti")	usr∉BD	Error	Sin Cambios
3	1,3,5	(7,usr,"Gana Atleti")	usr∈BD		Modifica la apuesta que había por el nuevo valor de apuesta
4	1,3,6	(5,usr,"Gana Atleti")	usr∈BD		Crea una nueva apuesta con el valor de 5

Valores límite

En el método createBet() deberemos calcular los valores límite para la cantidad con la que queremos apostar según el dinero que tengamos en el monedero. Para estudiar los VL deberemos crear 3 pruebas diferentes. Una con dinero de sobra, otra con el dinero justo y otra con un poco menos del dinero necesario para realizar una apuesta.

#	BD y Entrada	Resultado esperado
1	usr∈BD y tiene 51 € en el monedero (50,usr,"Gana Atleti")	Devolverá 1.0 que es el dinero restante que le quedará en el monedero.
2	usr∈BD y tiene 50 € en el monedero (50,usr,"Gana Atleti")	Devolverá 0.0 que es el dinero restante que le quedará en el monedero
2	usr∈BD y tiene 49 € en el monedero (50,usr,"Gana Atleti")	Saldrá una excepción avisando de que no hay saldo suficiente.

Defectos encontrados:

Los defectos han sido principalmente defectos de programación. En el proyecto inicial de Bets que hicimos el curso pasado deberíamos haber tratado más excepciones en los métodos de la clase DataAccess. En vez de tratarlos en estos métodos, se trataron en el GUI. No está de más tratarlos en el GUI, pero la clase más importante donde se deberían de tratar es en la clase que contacta con la BD para asegurar su correcto funcionamiento.

Pruebas de Integración:

Para las pruebas de integración, se ha creado la clase CreateBetBLBMTest.java donde se ha mockeado la clase DataAccess para poder realizar las pruebas.

Clases de equivalencia

Condición entrada	Clases de equivalencia válida	Clases de equivalencia no válida
Saldo suficiente	(monedero- apuesta) >=0 (1)	(monedero- apuesta) <0 (2)
Realizar puesta sin apuesta previa	findBet()== null (6)	

Valores límite

En cuanto a los valores límites, serán los mismos a los realizados en el estudio de caja negra.

#	BD y Entrada	Resultado esperado
1	usr∈BD y tiene 51 € en el monedero (50,usr,"Gana Atleti")	Devolverá 1.0 que es el dinero restante que le quedará en el monedero.
2	usr∈BD y tiene 50 € en el monedero (50,usr,"Gana Atleti")	Devolverá 0.0 que es el dinero restante que le quedará en el monedero
3	usr∈BD y tiene 49 € en el monedero (50,usr,"Gana Atleti")	Saldrá una excepción avisando de que no hay saldo suficiente.

createEvent(): Unai Artano

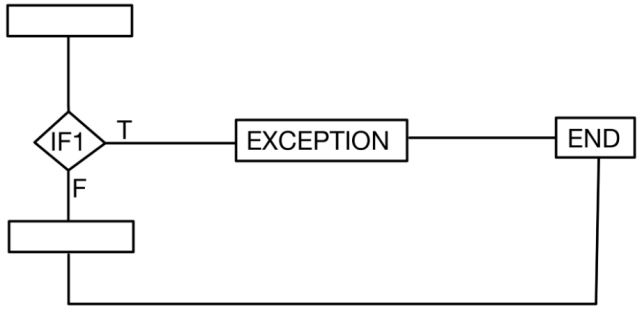
Código:

```
public Event createEvent(String description, Date eventDate) throws ObjectAlreadyExistException {
  System.out.println(">> DataAccess: CreateEvent => number = "+description+", date = "+eventDate);
  TypedQuery<Event> query = db.createQuery(
        "SELECT ev FROM Event ev WHERE ev.eventDate=?1 AND ev.description=?2", Event.class);
  query.setParameter(1, eventDate);
  query.setParameter(2, description);
  if(!query.getResultList().isEmpty()) throw new ObjectAlreadyExistException(
        "the event already exists");
  db.getTransaction().begin();
  Event evnt = new Event(description, eventDate);
  db.persist(evnt);
  db.getTransaction().commit();
  return evnt;
}
```

Pruebas Unitarias:

Caja Blanca:

Grafo:



Complejidad Ciclomática:

V(G)= #vértices- #nodos+ 2

$$V(G) = 5 - 5 + 2 = 2$$

V(G)= número de áreas cerradas+ 1

$$V(G) = 1 + 1 = 2$$

V(G)= #nodos condición+ 1

$$V(G) = 1 + 1 = 2$$

Caminos

Camino #1: IF1(T), EXCEPTION, END
Camino #2: IF1(F), END

Casos de prueba:

#	Camino	Cond. Entrada	Entrada		Resultado Esperado	
			Parámetros	Estado BD	Salida	Estado BD
1	IF1(T), END	evnt∈BD	(“Real Sociedad - Athletic Bilbao”, 2023/10/1)	e∈BD	Exception	Sin Cambio
2	IF(F), END	evnt∉BD	(“Real Sociedad - Athletic Bilbao”, 2024/2/22)	e∉BD	evnt	Crea un nuevo evento

```
public Event createEvent(String description, Date eventDate) throws ObjectAlreadyExistException {
    System.out.println(">> DataAccess: CreateEvent => number = " + description + ", date = " + eventDate);

    TypedQuery<Event> query = db.createQuery("SELECT ev FROM Event ev WHERE ev.eventDate=?1 AND ev.description=?2",
            Event.class);
    query.setParameter(1, eventDate);
    query.setParameter(2, description);
    if (!query.getResultList().isEmpty())
        throw new ObjectAlreadyExistException("the event already exists");

    db.getTransaction().begin();
    Event evnt = new Event(description, eventDate);
    db.persist(evnt);
    db.getTransaction().commit();
    return evnt;
}
```

Una vez que se hayan ejecutado los tests con JUnit, en la clase DataAccess.java en el método createEvent obtendremos una cobertura del 100%.

Caja Negra:

Clases de equivalencia:

Condición entrada	Clases de equivalencia válida	Clases de equivalencia no válida
Tiene que haber una fecha	Date != null (1)	Date = null (2)
Descripción del evento	descripción formato String (3.1) descripcion != “” (NULL) (3.2)	descripción formato cualquier otro que no sea string (4)
Evento exista en la BD	evnt∉BD (5)	evnt∈BD (6)

Casos de pruebas:

Nº casos	Clases de equivalencia cubiertas	Entrada		Resultado Esperado
		Parámetros	Estado BD	Salida
1	2	(“Real Sociedad - Athletic Bilbao”, null)		Error
2	4.1	(1, 2024/2/22)		Error
3	1, 3.1, 3.2,6	(“Real Sociedad - Athletic Bilbao”, 2023/10/1)	e∈BD	ObjectAlreadyExistException
4	1, 3.1, 3.2, 5	(“Real Sociedad - Athletic Bilbao”, 2024/2/22)	e∉BD	evnt

Valores límites:

Valores límites	Salida esperada
(“Real Sociedad - Athletic Bilbao”, 2023/2/22)	ObjectAlreadyExistException
(“Real Sociedad - Athletic Bilbao”, 2024/2/22)	evnt

Defectos encontrados:

Analizamos muchos más casos (de comprobaciones) en la clase principal (*CreateEventGUI.java*) que en la clase de *DataAccess.java*. Es un defecto que ya estaba desde el proyecto del año, pero ahora nos hemos dado cuenta que la clase principal no debe de darle demasiado trabajo; es decir, que parte de los casos que hemos analizado en la clase principal, deberíamos de haberlos analizado en la clase donde se sitúa la BD.

Pruebas de Integración:

Para realizar las pruebas, se ha mockeado la clase *DataAccess.java* para poder hacer hacer las pruebas de integración, y para ello se ha creado una clase nueva llamada *CreateEventBLBMTest.java*.

Clases de equivalencia

Condición entrada	Clases de equivalencia válida	Clases de equivalencia no válida
Crear evento	ev∉BD && ev.Date > fecha actual ("Real-Athletic, 2023/10/12")	ev∈BD
El evento ya se ha finalizado	ev.Date > fecha actual	ev.Date <= fecha actual

Valores límite

#	BD y Entrada	Resultado esperado
1	ev∈BD	ObjectAlreadyExistException
2	ev∉BD pero ya se ha terminado	("Real Sociedad - Athletic Bilbao", 2022/10/12)
3	ev∉BD y el evento no se ha terminado	("Real Sociedad - Athletic Bilbao", 2023/11/12)

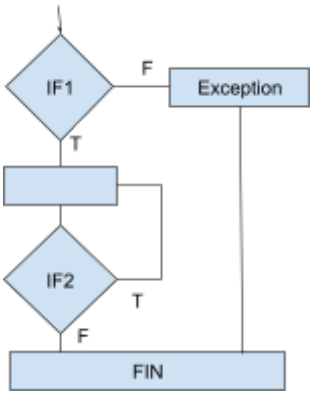
getEvents(): Martin Horsfield

Código:

Pruebas Unitarias:

Caja Blanca:

Grafo:



Aclaración: IF1 representa la excepción que puede saltar al acceder a la BD.

Complejidad Ciclomática:

V(G)= #vértices- #nodos+ 2

$$V(G) = 6 - 5 + 2 = 3$$

V(G)= número de áreas cerradas+ 1

$$V(G) = 2 + 1 = 3$$

V(G)= #nodos condición+ 1

$$V(G) = 2 + 1 = 3$$

Caja blanca:

#	Camino	Cond. Entrada	Entrada		Resultado Esperado	
			Parámetros	Estado BD	Salida	Estado BD
1	IF1(F)	date es null	(null)	date∉BD	Exception	Sin Cambio
2	IF1(F)	date es una fecha incorrecta	(“43/23/-1”) Fecha incorrecta	date∉BD	Exception	Sin Cambio
3	IF1(T), IF2(F)	ese día no tiene eventos	(“12/12/12”)	date∈BD	Vector<Events> vacio	Sin Cambio
4	IF1(T), IF2(T), IF2(F)	ese día tiene un evento	(“12/12/12”)	date∈BD	Vector<Events> un evento	Sin Cambio
5	IF1(T), IF2(T)xN , IF2(F)	ese día tiene n eventos	(“12/12/12”)	date∈BD	Vector<Events> n eventos	Sin Cambio

```
public Vector<Event> getEvents(Date date) {
    System.out.println(">> DataAccess: getEvents");
    Vector<Event> res = new Vector<Event>();
    TypedQuery<Event> query = db.createQuery("SELECT ev FROM Event ev WHERE ev.eventDate=?1", Event.class);
    query.setParameter(1, date);
    List<Event> events = query.getResultList();
    for (Event ev : events) {
        System.out.println(ev.toString());
        res.add(ev);
    }
    return res;
}
```

Caja Negra:

Clases de equivalencia:

Condición entrada	Clases de equivalencia válida	Clases de equivalencia no válida
Fecha nula	fecha != null (1)	fecha == null (2)
Fecha incorrecta	formato correcto fecha (3)	formato incorrecto fecha (4)
Fecha sin eventos	events.isEmpty()==false (5)	
Fecha con n eventos	events.length()>1 (6)	

Batería de casos de prueba

#	Clases de equivalencia cubiertas	Entrada		Resultado Esperado	
		Parámetros	Estado BD	Salida	Estado DB
1	2	(null)	N/A	Error	Sin Cambios
2	4	(Date)	N/A	Error	Sin Cambios
3	1,3,5	(Date)	events∉BD	Array<events>	Sin Cambios
4	1,3,6	(Date)	events∈BD	Array<events>	Sin Cambios

Pruebas de Integración:

Para realizar las pruebas, se ha mockeado la clase DataAccess.java para poder hacer hacer las pruebas de integración, y para ello se ha creado una clase nueva llamada GetEventsBLBMTest.java.

Clases de equivalencia

Condición entrada	Clases de equivalencia válida	Clases de equivalencia no válida
una fecha sin eventos	5	
una fecha con n eventos	6	

Valores límite

#	BD y Entrada	Resultado esperado
una fecha con solo 1 evento	fecha∈BD && events.size()==1	Ejecución correcta y devolución de “events”
una fecha con más de un evento	fecha∈BD && events.size()>1	Ejecución correcta y devolución de “events”

Conclusión

En este proyecto de ingeniería de software, hemos aplicado herramientas y técnicas esenciales como SonarCloud, pruebas de caja negra y caja blanca, Mockito, JUnit y refactorización. Hemos aprendido que la calidad del código, pruebas exhaustivas y la refactorización continua son cruciales para lograr software fiable y de alta calidad. Estas habilidades son fundamentales tanto en entornos académicos como en la industria.

Además, hemos trabajado con herramientas actuales que probablemente también usaremos en el futuro mundo laboral como SonarCloud y GitHub, que hasta ahora no habíamos usado ni teníamos conocimiento de ellas.