DOCUMENTACIÓN PRUEBAS SOFTWARE RIDES24COMPLETE

Ingeniería de Software II
2024-25

Jon Ander Iturrioz

RECURSOS

- Repositorio de GitHub:
    - https://github.com/JonAnderIturrioz/Rides24Complete

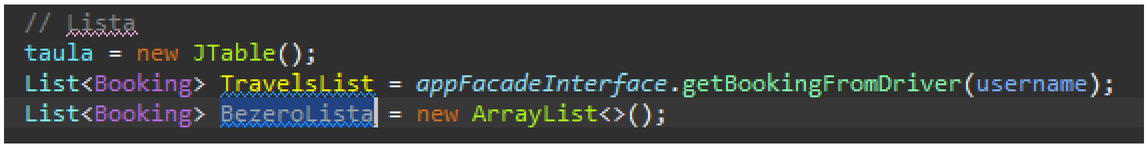
- Proyecto de SonarCloud:
    - https://sonarcloud.io/project/overview?id=rides24

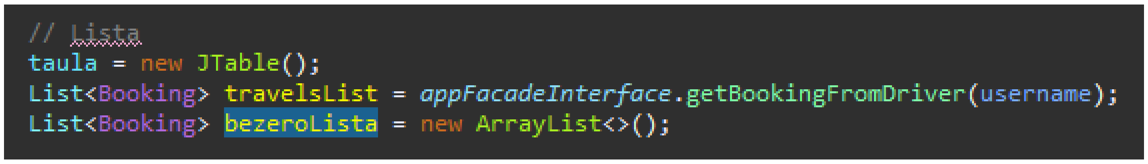
- Carpeta con imágenes (en caso de que no se vean bien):
    - https://drive.google.com/drive/folders/1QIKe972ytL0g6QEKP6EHSX9u5PfJpGkd?usp=sharing

ISSUES:

CONSISTENCY

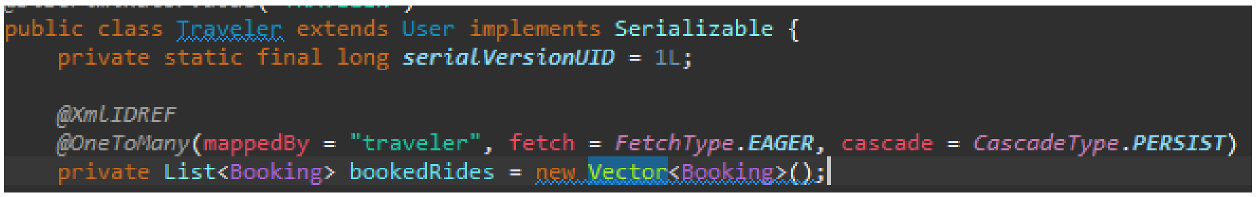
- **Minor**

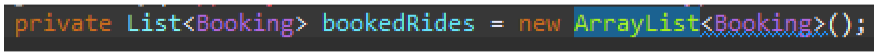

```
// Lista
taula = new JTable();
List<Booking> TravelsList = appFacadeInterface.getBookingFromDriver(username);
List<Booking> BezeroLista = new ArrayList<>();
```


```
// Lista
taula = new JTable();
List<Booking> travelsList = appFacadeInterface.getBookingFromDriver(username);
List<Booking> bezeroLista = new ArrayList<>();
```

Se han renombrado las listas indicadas para que sean consistentes con las demás variables.

- **Major**


```
public class Traveler extends User implements Serializable {
    private static final long serialVersionUID = 1L;

    @XmlIDREF
    @OneToMany(mappedBy = "traveler", fetch = FetchType.EAGER, cascade = CascadeType.PERSIST)
    private List<Booking> bookedRides = new Vector<Booking>();
```


```
private List<Booking> bookedRides = new ArrayList<Booking>();
```

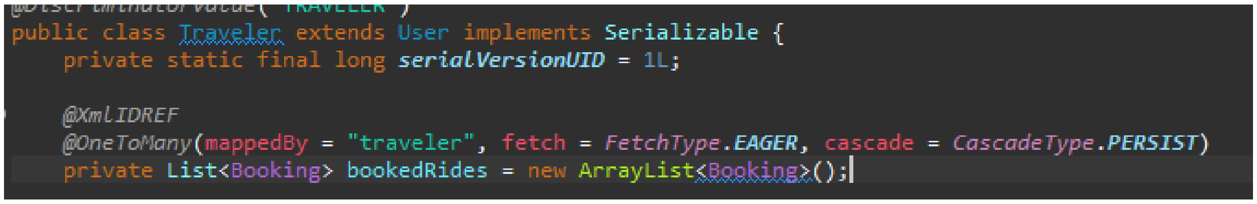
El tipo de lista ha pasado de Vector a ArrayList porque, siendo Vector un tipo de objeto sincronizado, tiene un mayor impacto en el rendimiento del programa.

- **Critical**

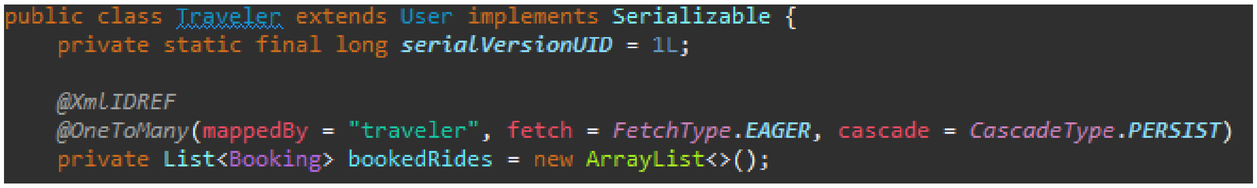
No se ha encontrado ningún error de este tipo en el código.

INTENTIONALITY

- **Minor**



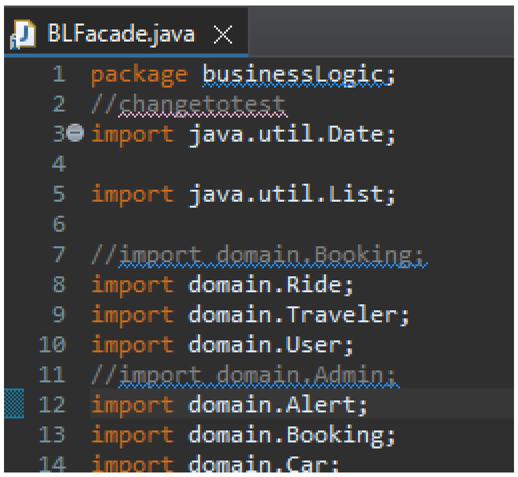
```
@DiscriminatorValue("TRAVELER")
public class Traveler extends User implements Serializable {
    private static final long serialVersionUID = 1L;

    @XmlIDREF
    @OneToMany(mappedBy = "traveler", fetch = FetchType.EAGER, cascade = CascadeType.PERSIST)
    private List<Booking> bookedRides = new ArrayList<Booking>();
```



```
public class Traveler extends User implements Serializable {
    private static final long serialVersionUID = 1L;

    @XmlIDREF
    @OneToMany(mappedBy = "traveler", fetch = FetchType.EAGER, cascade = CascadeType.PERSIST)
    private List<Booking> bookedRides = new ArrayList<>();
```

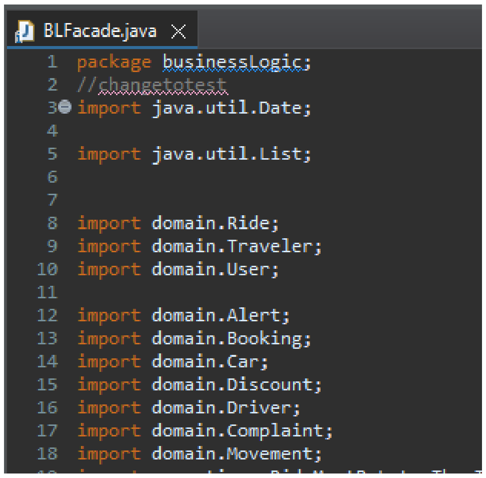
El compilador puede inferir el tipo de objeto que la lista debe guardar (Java 7 o más reciente), así que se elimina la segunda mención para reducir la verbosidad del código.

- **Major**



BLFacade.java

```
package businessLogic;
//changetotest
import java.util.Date;

import java.util.List;

//import domain.Booking;
import domain.Ride;
import domain.Traveler;
import domain.User;
//import domain.Admin;
import domain.Alert;
import domain.Booking;
import domain.Car;
```

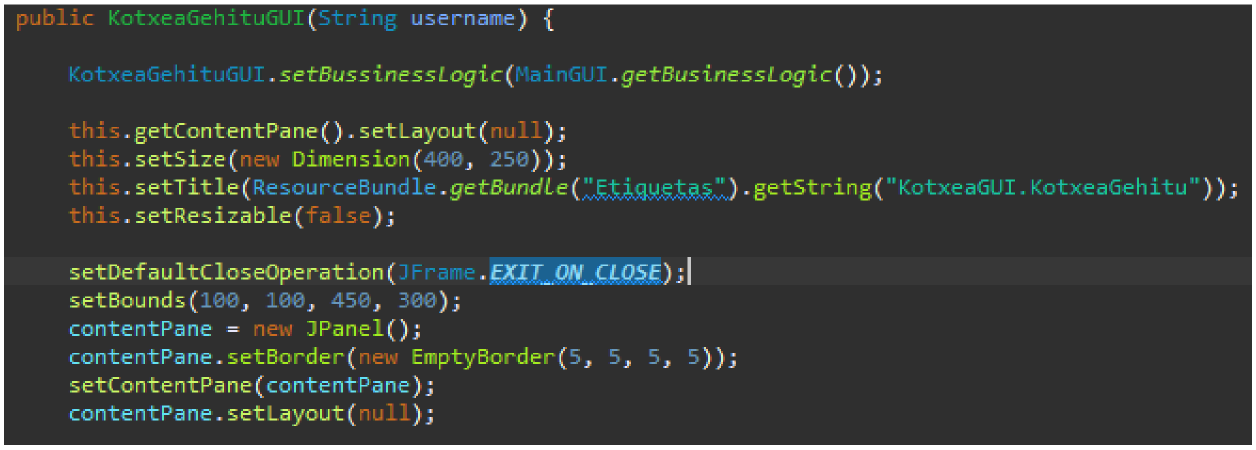


BLFacade.java

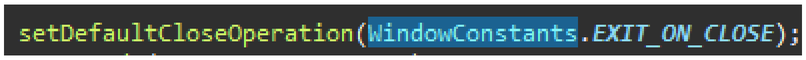
```
package businessLogic;
//changetotest
import java.util.Date;

import java.util.List;


import domain.Ride;
import domain.Traveler;
import domain.User;

import domain.Alert;
import domain.Booking;
import domain.Car;
import domain.Discount;
import domain.Driver;
import domain.Complaint;
import domain.Movement;
```

Se han eliminado imports comentados, ya que están en desuso y no son necesarios.

- **Critical**



```
public KotxeaGehituGUI(String username) {

    KotxeaGehituGUI.setBussinessLogic(MainGUI.getBusinessLogic());

    this.getContentPane().setLayout(null);
    this.setSize(new Dimension(400, 250));
    this.setTitle(ResourceBundle.getBundle("Etiquetas").getString("KotxeaGUI.KotxeaGehitu"));
    this.setResizable(false);

    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 450, 300);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    setContentPane(contentPane);
    contentPane.setLayout(null);
```

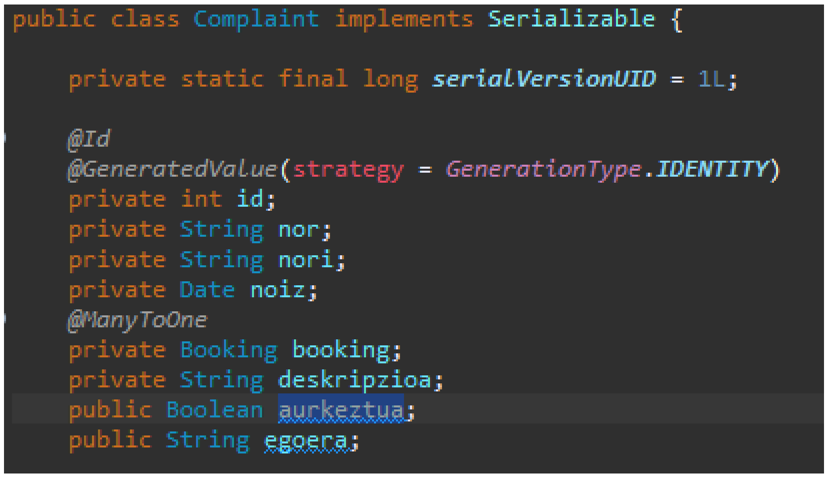


```
setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
```

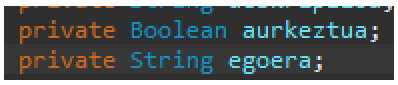
JFrame implementa funciones de WindowConstants. Dado que EXIT_ON_CLOSE es un miembro estático asociado a la clase, es preferible utilizar WindowsConstants directamente.

ADAPTABILITY

- **Minor**



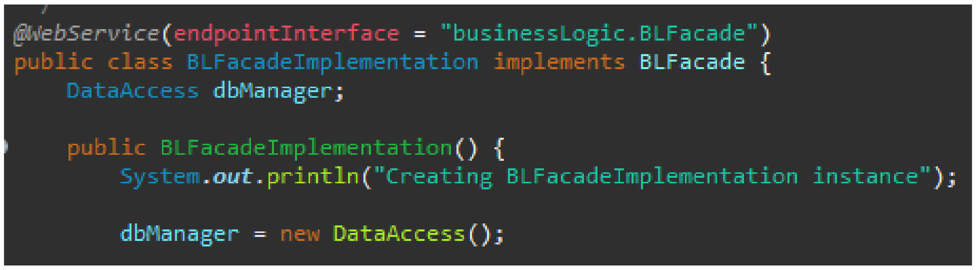
```java
public class Complaint implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String nor;
    private String nori;
    private Date noiz;
    @ManyToOne
    private Booking booking;
    private String deskripzioa;
    public Boolean aurkeztua;
    public String egoera;
```



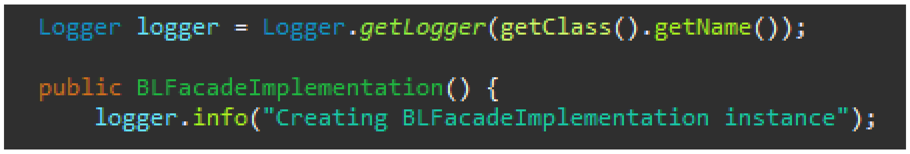
```java
private Boolean aurkeztua;
private String egoera;
```

Se ha cambiado la accesibilidad de los campos indicados a privado para prevenir cambios no autorizados a los valores.

- **Major**



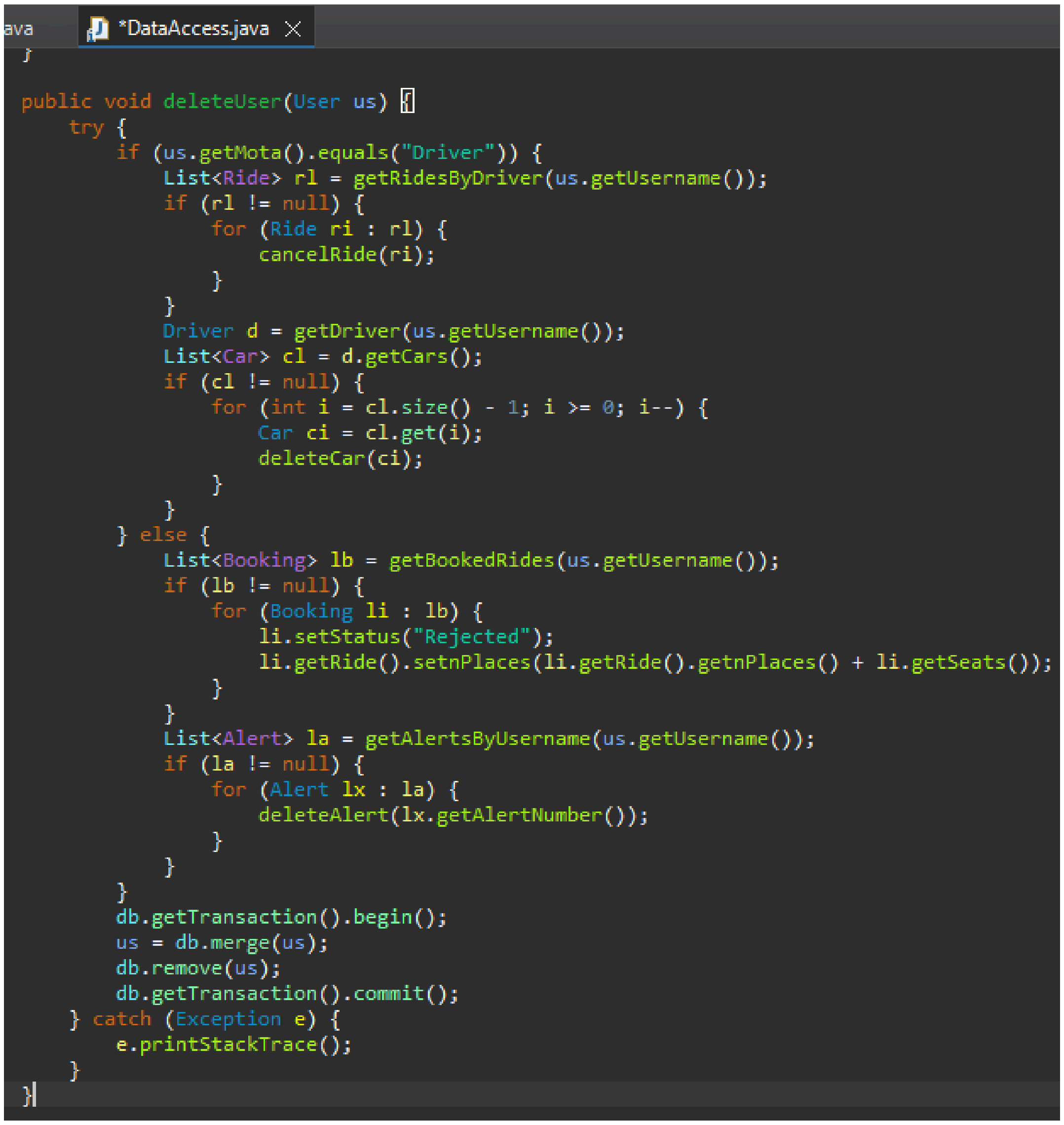
```java
@WebService(endpointInterface = "businessLogic.BLFacade")
public class BLFacadeImplementation implements BLFacade {
    DataAccess dbManager;

    public BLFacadeImplementation() {
        System.out.println("Creating BLFacadeImplementation instance");

        dbManager = new DataAccess();
```



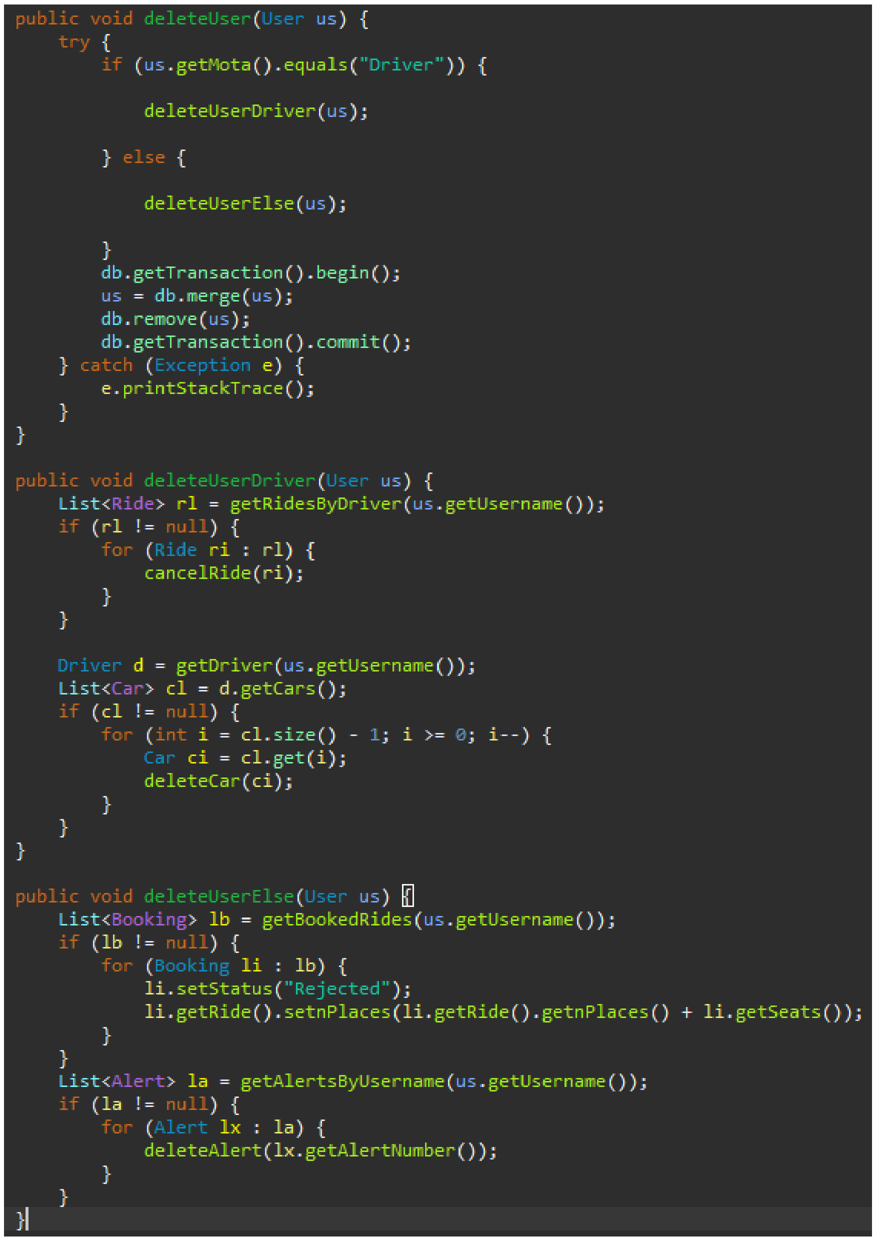
```java
Logger logger = Logger.getLogger(getClass().getName());

public BLFacadeImplementation() {
    logger.info("Creating BLFacadeImplementation instance");
```

Se recomienda utilizar logger en vez de System.out.println para que los mensajes se registren en logs de ejecución.

- **Critical**


ava *DataAccess.java

```
}

public void deleteUser(User us) {
    try {
        if (us.getMota().equals("Driver")) {
            List<Ride> rl = getRidesByDriver(us.getUsername());
            if (rl != null) {
                for (Ride ri : rl) {
                    cancelRide(ri);
                }
            }
            Driver d = getDriver(us.getUsername());
            List<Car> cl = d.getCars();
            if (cl != null) {
                for (int i = cl.size() - 1; i >= 0; i--) {
                    Car ci = cl.get(i);
                    deleteCar(ci);
                }
            }
        } else {
            List<Booking> lb = getBookedRides(us.getUsername());
            if (lb != null) {
                for (Booking li : lb) {
                    li.setStatus("Rejected");
                    li.getRide().setnPlaces(li.getRide().getnPlaces() + li.getSeats());
                }
            }
            List<Alert> la = getAlertsByUsername(us.getUsername());
            if (la != null) {
                for (Alert lx : la) {
                    deleteAlert(lx.getAlertNumber());
                }
            }
        }
        db.getTransaction().begin();
        us = db.merge(us);
        db.remove(us);
        db.getTransaction().commit();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```



```java
public void deleteUser(User us) {
    try {
        if (us.getMota().equals("Driver")) {

            deleteUserDriver(us);

        } else {

            deleteUserElse(us);

        }
        db.getTransaction().begin();
        us = db.merge(us);
        db.remove(us);
        db.getTransaction().commit();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public void deleteUserDriver(User us) {
    List<Ride> rl = getRidesByDriver(us.getUsername());
    if (rl != null) {
        for (Ride ri : rl) {
            cancelRide(ri);
        }
    }

    Driver d = getDriver(us.getUsername());
    List<Car> cl = d.getCars();
    if (cl != null) {
        for (int i = cl.size() - 1; i >= 0; i--) {
            Car ci = cl.get(i);
            deleteCar(ci);
        }
    }
}

public void deleteUserElse(User us) {
    List<Booking> lb = getBookedRides(us.getUsername());
    if (lb != null) {
        for (Booking li : lb) {
            li.setStatus("Rejected");
            li.getRide().setnPlaces(li.getRide().getnPlaces() + li.getSeats());
        }
    }
    List<Alert> la = getAlertsByUsername(us.getUsername());
    if (la != null) {
        for (Alert lx : la) {
            deleteAlert(lx.getAlertNumber());
        }
    }
}
```

El método deleteUser de DataAccess tenía 23 líneas de código. Dado que la longitud máxima recomendada de un módulo es de 15, lo he separado en 3 métodos diferentes que llevan a cabo la misma función en conjunto.

RESPONSIBILITY

- **Minor**

No se ha encontrado ningún error de este tipo en el código.

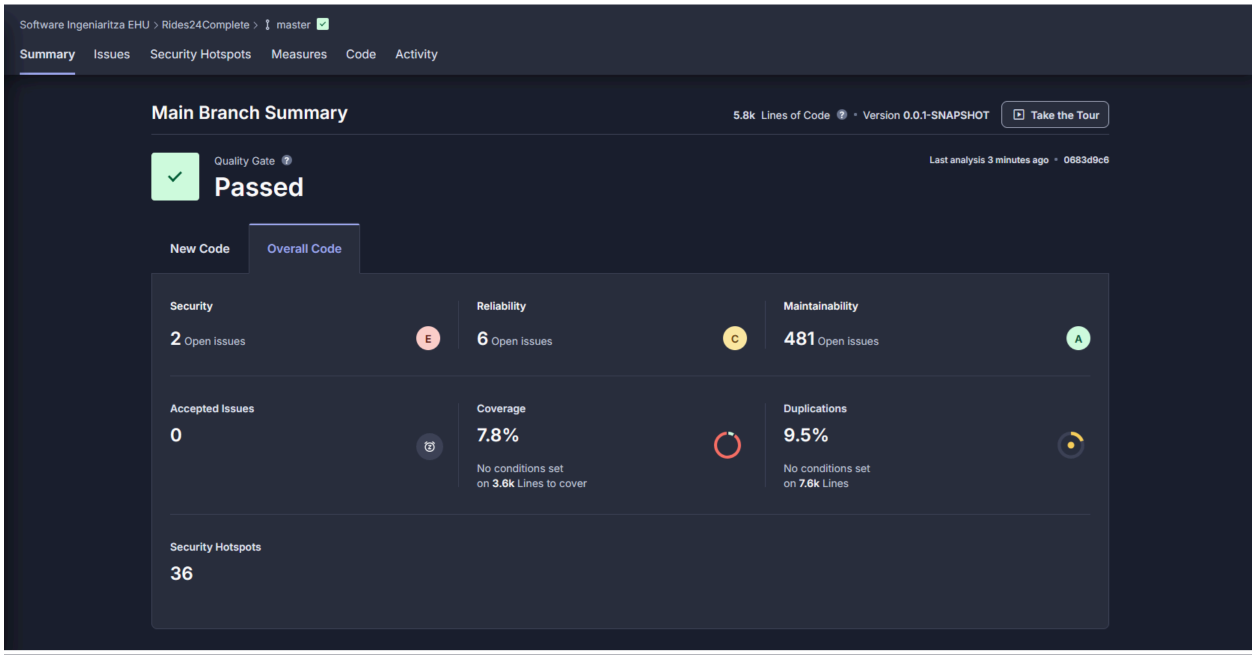
- **Major**

No se ha encontrado ningún error de este tipo en el código.

- **Critical**

No se ha encontrado ningún error de este tipo en el código.

INFORMES SONARCLOUD


Software Ingeniaritza EHU > Rides24Complete > master

Summary | Issues | Security Hotspots | Measures | Code | Activity

Main Branch Summary

5.8k Lines of Code • Version 0.0.1-SNAPSHOT | Take the Tour

Quality Gate
Passed

Last analysis 3 minutes ago • 0683d9c6

New Code | Overall Code

Security
2 Open issues — E

Reliability
6 Open issues — C

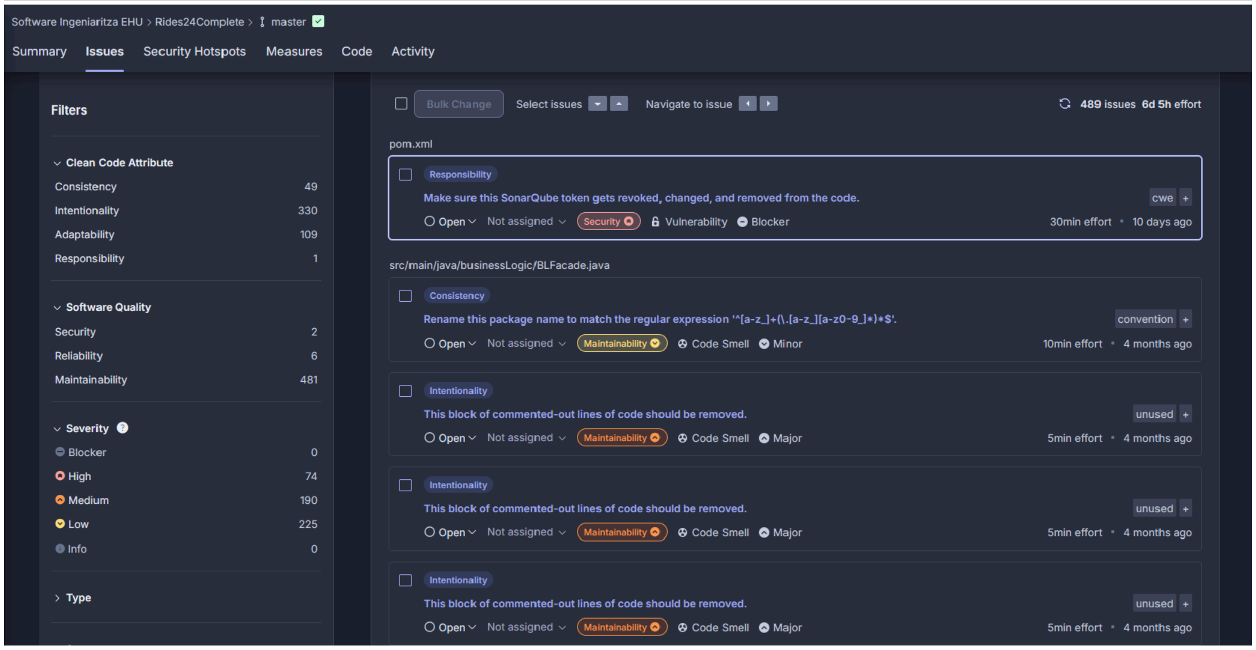
Maintainability
481 Open issues — A

Accepted Issues
0

Coverage
7.8%
No conditions set
on 3.6k Lines to cover

Duplications
9.5%
No conditions set
on 7.6k Lines

Security Hotspots
36


Software Ingeniaritza EHU > Rides24Complete > master

Summary | Issues | Security Hotspots | Measures | Code | Activity

Filters

Clean Code Attribute

Consistency	49
Intentionality	330
Adaptability	109
Responsibility	1

Software Quality

Security	2
Reliability	6
Maintainability	481

Severity

Blocker	0
High	74
Medium	190
Low	225
Info	0

Type

Bulk Change | Select issues | Navigate to issue | 489 issues 6d 5h effort

pom.xml

Responsibility
Make sure this SonarQube token gets revoked, changed, and removed from the code. — cwe
Open · Not assigned · Security · Vulnerability · Blocker — 30min effort • 10 days ago

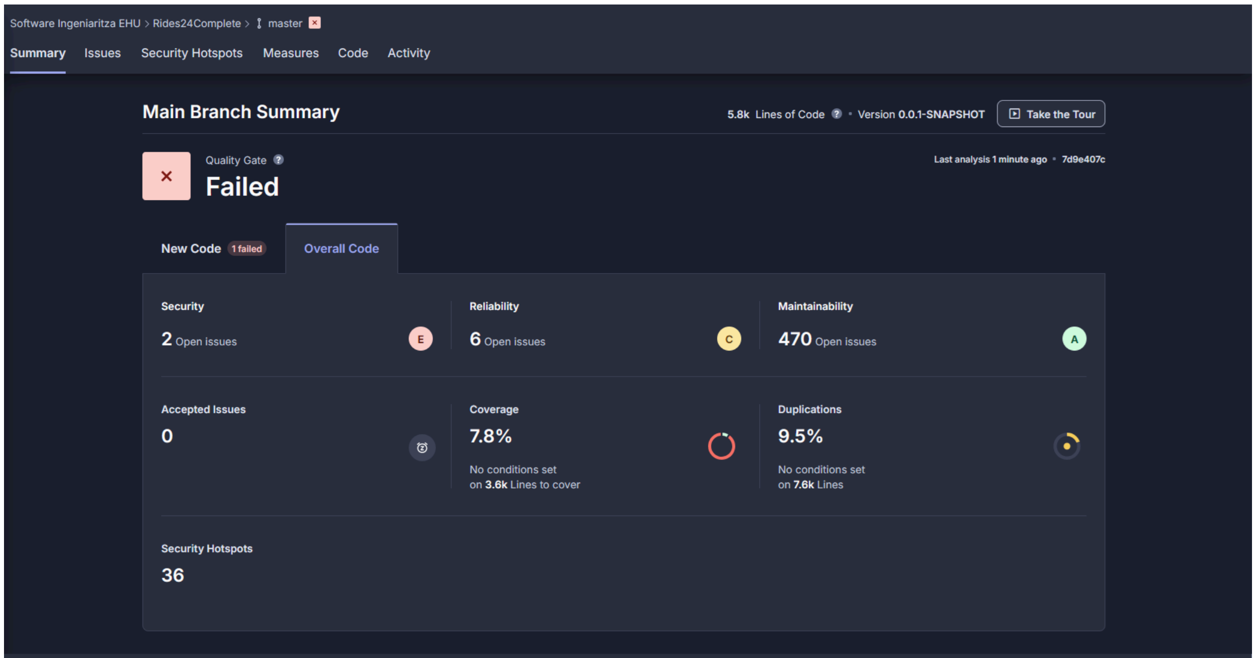
src/main/java/businessLogic/BLFacade.java

Consistency
Rename this package name to match the regular expression '^[a-z_]+(\.[a-z_][a-z0-9_]*)*$'. — convention
Open · Not assigned · Maintainability · Code Smell · Minor — 10min effort • 4 months ago

Intentionality
This block of commented-out lines of code should be removed. — unused
Open · Not assigned · Maintainability · Code Smell · Major — 5min effort • 4 months ago

Intentionality
This block of commented-out lines of code should be removed. — unused
Open · Not assigned · Maintainability · Code Smell · Major — 5min effort • 4 months ago

Intentionality
This block of commented-out lines of code should be removed. — unused
Open · Not assigned · Maintainability · Code Smell · Major — 5min effort • 4 months ago


Software Ingeniaritza EHU > Rides24Complete > master

Summary Issues Security Hotspots Measures Code Activity

Main Branch Summary

5.8k Lines of Code • Version 0.0.1-SNAPSHOT | Take the Tour

Quality Gate

Failed

Last analysis 1 minute ago • 7d9e407c

New Code 1 failed | Overall Code

Security: 2 Open issues (E)

Reliability: 6 Open issues (C)

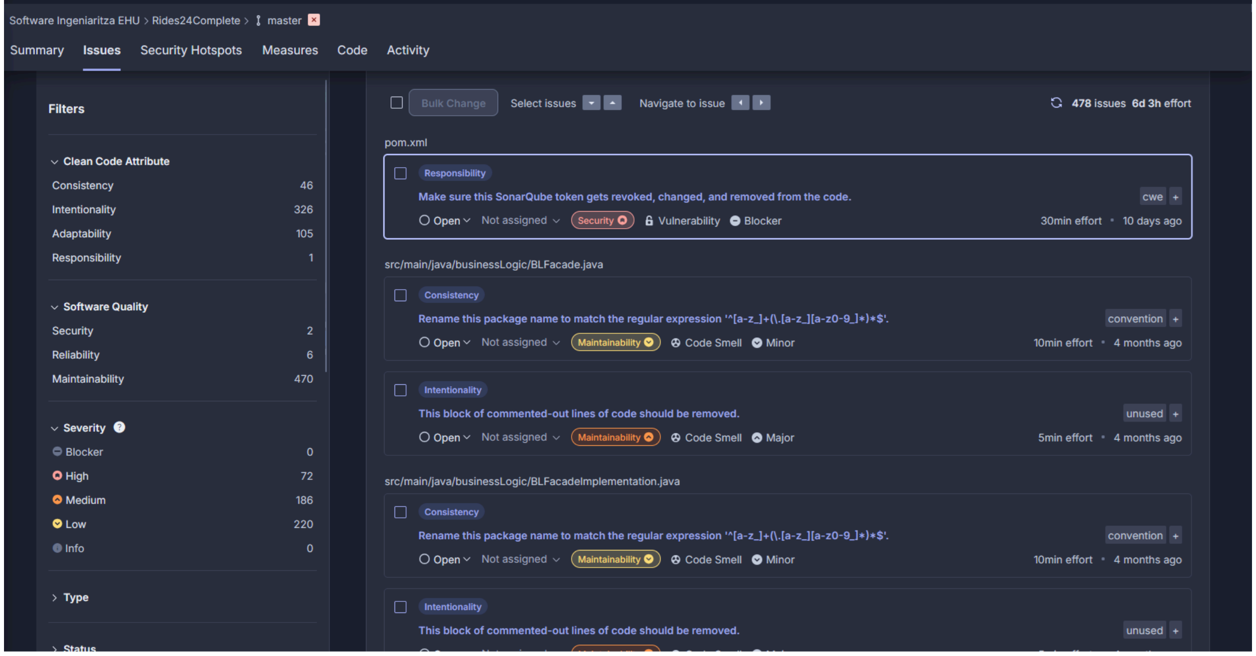
Maintainability: 470 Open issues (A)

Accepted Issues: 0

Coverage: 7.8%
No conditions set
on 3.6k Lines to cover

Duplications: 9.5%
No conditions set
on 7.6k Lines

Security Hotspots: 36


Software Ingeniaritza EHU > Rides24Complete > master

Summary Issues Security Hotspots Measures Code Activity

Filters

Clean Code Attribute

Consistency	46
Intentionality	326
Adaptability	105
Responsibility	1

Software Quality

Security	2
Reliability	6
Maintainability	470

Severity

Blocker	0
High	72
Medium	186
Low	220
Info	0

Type

Status

Bulk Change | Select issues | Navigate to issue | 478 issues 6d 3h effort

pom.xml

Responsibility
Make sure this SonarQube token gets revoked, changed, and removed from the code. cwe
Open Not assigned Security Vulnerability Blocker 30min effort • 10 days ago

src/main/java/businessLogic/BLFacade.java

Consistency
Rename this package name to match the regular expression '^[a-z_]+(\.[a-z_][a-z0-9_]*)*$'. convention
Open Not assigned Maintainability Code Smell Minor 10min effort • 4 months ago

Intentionality
This block of commented-out lines of code should be removed. unused
Open Not assigned Maintainability Code Smell Major 5min effort • 4 months ago

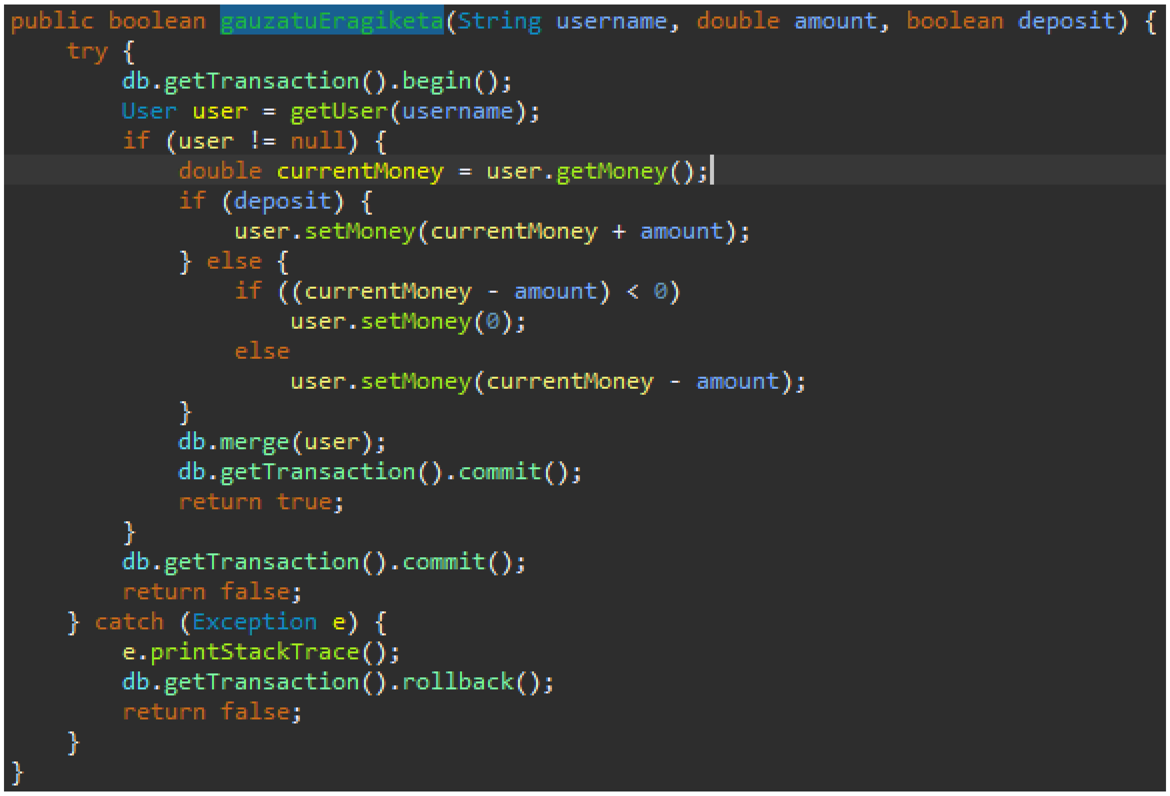
src/main/java/businessLogic/BLFacadeImplementation.java

Consistency
Rename this package name to match the regular expression '^[a-z_]+(\.[a-z_][a-z0-9_]*)*$'. convention
Open Not assigned Maintainability Code Smell Minor 10min effort • 4 months ago

Intentionality
This block of commented-out lines of code should be removed. unused

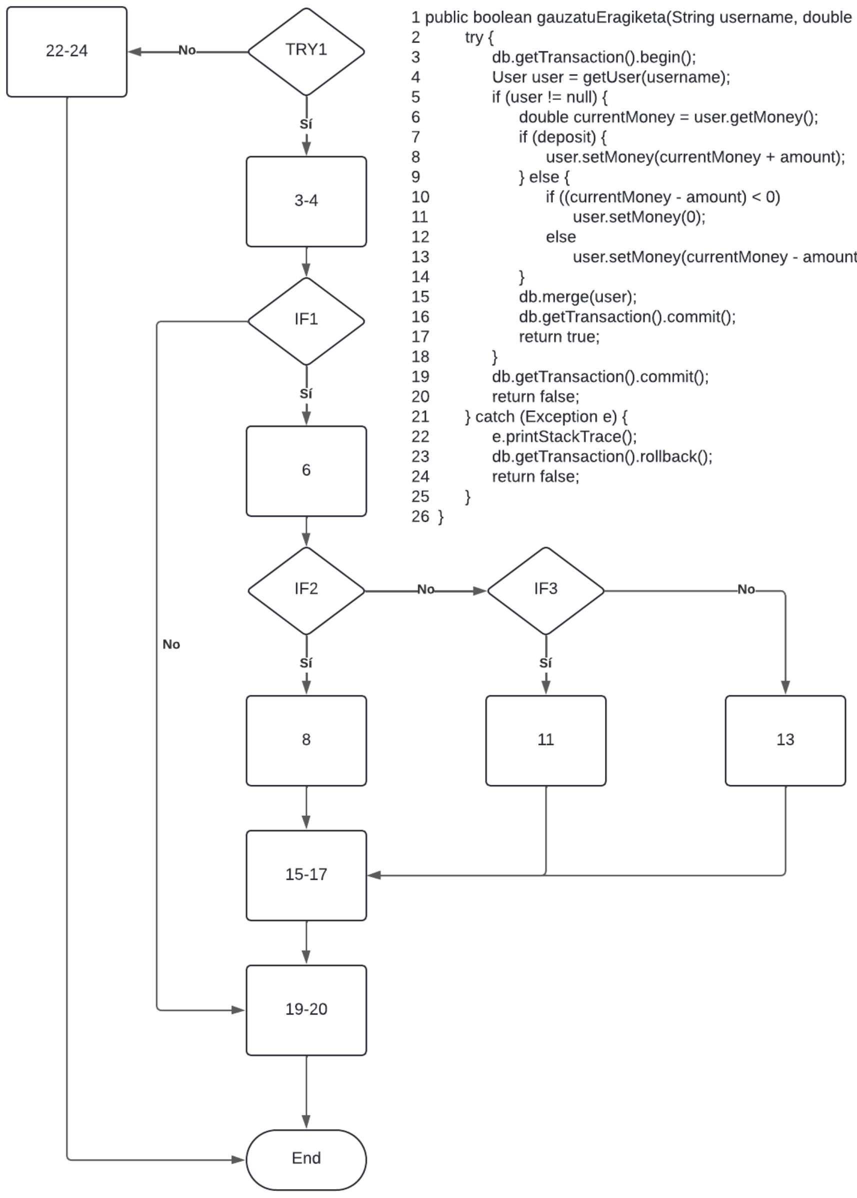
TESTS

CÓDIGO DEL MÉTODO ELEGIDO:



```java
public boolean gauzatuEragiketa(String username, double amount, boolean deposit) {
    try {
        db.getTransaction().begin();
        User user = getUser(username);
        if (user != null) {
            double currentMoney = user.getMoney();
            if (deposit) {
                user.setMoney(currentMoney + amount);
            } else {
                if ((currentMoney - amount) < 0)
                    user.setMoney(0);
                else
                    user.setMoney(currentMoney - amount);
            }
            db.merge(user);
            db.getTransaction().commit();
            return true;
        }
        db.getTransaction().commit();
        return false;
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
        return false;
    }
}
```

DISENO CAJA BLANCA:



```
1 public boolean gauzatuEragiketa(String username, double amount, boolean deposit) {
2         try {
3                 db.getTransaction().begin();
4                 User user = getUser(username);
5                 if (user != null) {
6                         double currentMoney = user.getMoney();
7                         if (deposit) {
8                                 user.setMoney(currentMoney + amount);
9                         } else {
10                                if ((currentMoney - amount) < 0)
11                                        user.setMoney(0);
12                                else
13                                        user.setMoney(currentMoney - amount);
14                        }
15                        db.merge(user);
16                        db.getTransaction().commit();
17                        return true;
18                }
19                db.getTransaction().commit();
20                return false;
21        } catch (Exception e) {
22                e.printStackTrace();
23                db.getTransaction().rollback();
24                return false;
25        }
26 }
```

#	Camino	Condición	Entrada	BD	Resultado esperado	BD
1	TRY1(T) 3-4 IF1(F) 19-20 End	busqueda de usuario en db resulta en valor null	username="testuser"; amount= 10; deposit= true;	vacío	FALSE	vacío
2	TRY1(T) 3-4 IF1(T) 6 IF2(T) 8 15-17 19-20 End	la cantidad se deposita correctamente	username="testuser"; amount= 10; deposit= true;	[nombre=testuser, dinero=10]	TRUE	[nombre=testuser, dinero=20]
3	TRY1(T) 3-4 IF1(T) 6 IF2(F) IF3(T) 11 15-17 19-20 End	se quiere sacar mas de lo que tiene el usuario, el resultado es 0	username = "testuser"; amount = 10; deposit = false;	[nombre=testuser, dinero=5]	TRUE	[nombre=testuser, dinero=0]
4	TRY1(T) 3-4 IF1(T) 6 IF2(F) IF3(F) 13 15-17 19-20 End	se quiere sacar menos de lo que tiene el usuario, el resultado es cantidad antigua - cantidad a sacar	username = "testuser"; amount = 10; deposit = false;	[nombre=testuser, dinero=20]	TRUE	[nombre=testuser, dinero=10]
5	TRY1(F) 22-24 End	salta algún tipo de Exception que hace saltar el catch.	username = null; amount = 10; deposit = false;	[nombre=testuser, dinero=10]	FALSE	[nombre=testuser, dinero=10]

DISEÑO CAJA NEGRA:

	Condición	Clase de equvalencia válida	Clase de equvalencia no válida
Condición de entrada	usuario u está en BD	u ∈ BD (1)	u ∉ BD (2)
	nombre usuario alfanumérico	username es del tipo string (3)	username no es del tipo string (4)
	tamaño nombre usuario mayor que cero	username.length > 0 (5)	username.length = 0 (6)
	amount es real	amount tipo double (7)	amount es alfanumérico (8)
	amount es positivo	amount > 0 (9)	amount < 0 (10) amount = 0 (11)
	deposit es booleano	deposit==true o deposit ==false (12)	deposit!=true y deposit !=false (13)
	username no es null	username!=null (14)	u==null (15)
	amount no es null	amount!=null (16)	symptoms==null (17)
	deposit no es null	deposit!=null (18)	weights==null (19)
Comportamiento del programa	deposit es true	deposit == true (20)	
	deposit es false	deposit == false (21)	
	sacar menos de lo que tiene el usuario	u.getMoney() > amount (22)	
	sacar más de lo que tiene el usuario	u.getMoney() < amount (23)	
	sacar lo que tiene el usuario	u.getMoney() = amount (24)	

	Entrada	Estado BD	Clases de equivalencia cubiertas	Resultado esperado	Estado nuevo BD
P1	username = "testuser" amount = 10 deposit = true	["testuser",...,0 ,...]	1, 3, 5, 7, 9, 12, 14, 16, 18, 20	TRUE	["testuser",...,10 ,...]
P2	username = "testuser" amount = 5 deposit = false	["testuser",...,10 ,...]	1, 3, 5, 7, 9, 12, 14, 16, 18, 21, 22	TRUE	["testuser",...,5 ,...]
P3	username = "testuser" amount = 15 deposit = false	["testuser",...,10 ,...]	1, 3, 5, 7, 9, 12, 14, 16, 18, 21, 23	TRUE	["testuser",...,0 ,...]
P4	username = "testuser" amount = 10 deposit = false	["testuser",...,10 ,...]	1, 3, 5, 7, 9, 12, 14, 16, 18, 21, 24	TRUE	["testuser",...,0 ,...]

N1	username = "testuser" amount = 10 deposit = true	vacío	2		FALSE	vacío	
~~N2~~	username = 12345 amount = 5 deposit = true	["testuser",...,0 ,...]	4		FALSE	["testuser",...,0 ,...]	Las funciones en java no aceptan datos de tipos incorrectos, no puedo testear esto
N3	username = "" amount = 5 deposit = true	["testuser",...,0 ,...]	6		FALSE	["testuser",...,0 ,...]	
~~N4~~	username = "testuser" amount = "abc" deposit = true	["testuser",...,0 ,...]	8		FALSE	["testuser",...,0 ,...]	Las funciones en java no aceptan datos de tipos incorrectos, no puedo testear esto
N5	username = "testuser" amount = -10 deposit = true	["testuser",...,10 ,...]	10		FALSE	["testuser",...,10 ,...]	
N6	username = "testuser" amount = 0 deposit = true	["testuser",...,0 ,...]	11		FALSE	["testuser",...,0 ,...]	
~~N7~~	username = "testuser" amount = 5 deposit = 5	["testuser",...,0 ,...]	13		FALSE	["testuser",...,0 ,...]	Las funciones en java no aceptan datos de tipos incorrectos, no puedo testear esto
N8	username = null amount = 5 deposit = true	["testuser",...,0 ,...]	15		FALSE	["testuser",...,0 ,...]	
N9	username = "testuser" amount = null deposit = true	["testuser",...,0 ,...]	17		FALSE	["testuser",...,0 ,...]	
N10	username = "testuser" amount = 10 deposit = null	["testuser",...,0 ,...]	19		FALSE	["testuser",...,0 ,...]	

FALLOS ENCONTRADOS

- El método acepta strings vacíos “” como nombre de usuario.
  - GauzatuEragiketaBDBlackTest negativeTest3
  - GauzatuEragiketaMockBlackTest negativeTest3

- El método acepta strings compuestos únicamente por números, pero no se si debe. Puede que este comportamiento sea aceptable.
  - GauzatuEragiketaMockBlackTest negativeTest2
  - Este test no está implementado en BD.

- El método acepta números negativos en el parámetro “amount” y procede con los cálculos.
  - GauzatuEragiketaBDBlackTest negativeTest5
  - GauzatuEragiketaMockBlackTest negativeTest5

- El método acepta el número 0 en el parámetro “amount”, y lleva a cabo una operación completamente redundante.
  - GauzatuEragiketaBDBlackTest negativeTest6
  - GauzatuEragiketaMockBlackTest negativeTest6

- El método lleva a cabo la operación si recibe un objeto User con nombre de usuario “null”, aunque hace falta forzarlo por mocks.
  - GauzatuEragiketaMockBlackTest negativeTest8
  - Es difícil analizar este comportamiento con la BD activa, ya que no es posible introducir un User con nombre de usuario “null”.

- El método lanza un NullPointerException al recibir el valor “null” en el parámetro amount. La excepción no se atrapa en el método.
  - GauzatuEragiketaBDBlackTest negativeTest9
  - GauzatuEragiketaMockBlackTest negativeTest9

- El método lanza un NullPointerException al recibir el valor “null” en el parámetro deposit. La excepción no se atrapa en el método.
  - GauzatuEragiketaBDBlackTest negativeTest10
  - GauzatuEragiketaMockBlackTest negativeTest10

NOTAS PARA EL PROFESOR:

- No he encontrado la manera de implementar los casos de prueba negativos propuestos N2 (parcial), N4 y N7 del análisis de caja negra, por lo que podría haber más problemas.
- El programa no recorre el camino del caso de prueba 1 del análisis de caja blanca en los tests con BD, ya que no he conseguido hacer que haya un User con nombre de usuario “null” en la BD.

GauzatuEragiketaBDBlackTest

```
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;
import static org.junit.Assert.fail;

import org.junit.Test;

import dataAccess.DataAccess;
import domain.Driver;
import testOperations.TestDataAccess;

public class GauzatuEragiketaBDBlackTest {
     //sut:system under test
     static DataAccess sut=new DataAccess();

     //additional operations needed to execute the test
     static TestDataAccess testDA=new TestDataAccess();

     private Driver driver;

     @Test
     public void positiveTest1() {
           String username ="testuser";
           String pass ="a";
           double amount = 10;
           boolean deposit = true;
           boolean driverCreated = false;


           try {
                 // Add testuser to database and save User object as "driver"
                 testDA.open();
                 if (!testDA.existDriver(username)) {
                 driver = testDA.createDriver(username,pass);
                 driverCreated = true;
                 } else driver = sut.getDriver(username);
```

```
			testDA.close();

			// Get expected money, current + amount to add
			// Should be 0 + 10 if new driver
			double expected = driver.getMoney() + amount;

			sut.open();
			// Starting amount of money is 0 because that's how new
// users are created
			// Run test
			boolean u=sut.gauzatuEragiketa(username, amount, deposit);

			// Get new money amount
			double current= sut.getDriver(username).getMoney();

			// Check function success and correct result
			assertTrue(u);
			assertEquals(expected , current, 0.001);

		}catch(Exception e) {
			e.printStackTrace();
			fail();
		} finally{
			// Cleanup
			testDA.open();
			if (driverCreated)
			testDA.removeDriver(username);
			testDA.close();
			sut.close();
		}

	}

	@Test
	public void positiveTest2() {
		String username ="testuser";
		String pass ="a";
		double amount = 5;
```

```
boolean deposit = false;
boolean driverCreated = false;


try {
      // Add testuser to database and save User object as "driver"
      testDA.open();
      if (!testDA.existDriver(username)) {
      driver = testDA.createDriver(username,pass);
      driverCreated = true;
      } else driver = sut.getDriver(username);
      testDA.close();

      // Get expected money, current + amount to add

      double expected = 10 - amount;

      sut.open();
      // Set money bigger than amount
      sut.gauzatuEragiketa(username, 10 , true);

      // Run test
      boolean u=sut.gauzatuEragiketa(username, amount, deposit);

      // Get new money amount
      double current= sut.getDriver(username).getMoney();

      // Check function success and correct result
      assertTrue(u);
      assertEquals(expected , current, 0.001);

}catch(Exception e) {
      e.printStackTrace();
      fail();
} finally{
      // Cleanup
      testDA.open();
      if (driverCreated)
```

```
			testDA.removeDriver(username);
			testDA.close();
			sut.close();
		}

	}

	@SuppressWarnings({ "unchecked" })
	@Test
	public void positiveTest3() {
		String username = "testuser";
		String pass ="a";
		double amount = 15;
		boolean deposit = false;
		boolean driverCreated = false;

		try {

			// Add testuser to database and save User object as "driver"
			testDA.open();
			if (!testDA.existDriver(username)) {
			driver = testDA.createDriver(username,pass);
			driverCreated = true;
			} else driver = sut.getDriver(username);
			testDA.close();

			// Expected money should be negative, which should be corrected
to 0
			double expected = 0;

			sut.open();

			// Set up quantity smaller than amount to run test
			sut.gauzatuEragiketa(username, 10 , true);

			// Run test
			boolean u=sut.gauzatuEragiketa(username, amount , deposit);
```

```
            // Get new money amount
            double current= sut.getDriver(username).getMoney();

            // Check function success and correct result
            assertTrue(u);
            assertEquals(expected , current, 0.001);

      }catch(Exception e) {
            e.printStackTrace();
            fail();
      } finally{
            // Cleanup
            testDA.open();
            if (driverCreated)
            testDA.removeDriver(username);
            testDA.close();
            sut.close();
      }

}

@Test
public void positiveTest4() {

      String username = "testuser";
      String pass ="a";
      double amount = 10;
      boolean deposit = false;
      boolean driverCreated = false;

      try {

            // Add testuser to database and save User object as "driver"
            testDA.open();
            if (!testDA.existDriver(username)) {
            driver = testDA.createDriver(username,pass);
            driverCreated = true;
            } else driver = sut.getDriver(username);
            testDA.close();

            sut.open();

            // Add money to driver for result zero
            sut.gauzatuEragiketa(username, amount , true);

            // Set expected result, which should be 0
            double expected = 0;

            // Run test
            boolean u=sut.gauzatuEragiketa(username, amount, deposit);

            // Get new money amount
            double current= sut.getDriver(username).getMoney();

            // Check function success and correct result
            assertTrue(u);
            assertEquals(expected , current, 0.001);

      }catch(Exception e) {
            e.printStackTrace();
            fail();
      } finally{
            // Cleanup
            testDA.open();
            if (driverCreated)
            testDA.removeDriver(username);
            testDA.close();
            sut.close();
      }

}

@Test
public void negativeTest1() {
      String username = "testuser"; //no object of this username in the DB
      double amount = 10;
```

```
		boolean deposit = true;

		try {


			sut.open();
			// Run test with empty DB
			boolean u=sut.gauzatuEragiketa(username, amount, deposit);

			// Check function success and correct result
			assertFalse(u);

		}catch(Exception e) {
			e.printStackTrace();
			fail();
		} finally{
			// Cleanup
			sut.close();
		}
	}

	@Test
	public void negativeTest3() {
		String username = "";
		String pass ="a";
		double amount = 10;
		boolean deposit = false;
		boolean driverCreated = false;

		try {

			// Add testuser to database and save User object as "driver"
			testDA.open();
			if (!testDA.existDriver(username)) {
			driver = testDA.createDriver(username,pass); // even this fuction
probably shouldn't accept empty strings...
			driverCreated = true;
			} else driver = sut.getDriver(username);
```

```
            testDA.close();

            sut.open();

            // Run test
            boolean u=sut.gauzatuEragiketa(username, amount, deposit);


            // Should likely either fail or throw an exception, does neither.
            assertFalse(u);

      }catch(Exception e) {
            e.printStackTrace();
            fail();
      } finally{
            // Cleanup
            testDA.open();
            if (driverCreated)
            testDA.removeDriver(username);
            testDA.close();
            sut.close();
      }
}

@Test
public void negativeTest5() {
      String username = "testuser";
      String pass ="a";
      double amount = -10;
      boolean deposit = true;
      boolean driverCreated = false;

      try {

            // Add testuser to database and save User object as "driver"
            testDA.open();
            if (!testDA.existDriver(username)) {
            driver = testDA.createDriver(username,pass);
```

```
                driverCreated = true;
                } else driver = sut.getDriver(username);
                testDA.close();

                sut.open();
                // Set baseline amount
                sut.gauzatuEragiketa(username, amount, true);


                // Run test
                boolean u=sut.gauzatuEragiketa(username, amount, deposit);

                // Check if operation is done with non-valid numbers
                // If operation goes through, baseline amount should change and
lead to failure
                assertEquals(10, sut.getActualMoney(username), 0.001);

                // Negative numbers should not be accepted.
                // It defeats the point of the deposit parameter.
                assertFalse(u);

          }catch(Exception e) {
                e.printStackTrace();
                fail();
          } finally{
                // Cleanup
                testDA.open();
                if (driverCreated)
                testDA.removeDriver(username);
                testDA.close();
                sut.close();
          }
     }

     @Test
     public void negativeTest6() {
          String username = "testuser";
          String pass ="a";
```

```
		double amount = 0;
		boolean deposit = true;
		boolean driverCreated = false;

		try {

			// Add testuser to database and save User object as "driver"
			testDA.open();
			if (!testDA.existDriver(username)) {
			driver = testDA.createDriver(username,pass);
			driverCreated = true;
			} else driver = sut.getDriver(username);
			testDA.close();

			sut.open();

			// Run test
			boolean u=sut.gauzatuEragiketa(username, amount, deposit);

			// Operation with amount 0 is redundant and should not be
accepted.
			assertFalse(u);

		}catch(Exception e) {
			e.printStackTrace();
			fail();
		} finally{
			// Cleanup
			testDA.open();
			if (driverCreated)
			testDA.removeDriver(username);
			testDA.close();
			sut.close();
		}
	}

	@Test
	public void negativeTest8() {
		String username = null;
		//String pass = "a";
		double amount = 10;
		boolean deposit = true;
		//boolean driverCreated = false;

		try {
			// Add testuser to database and save User object as "driver"
			/*testDA.open();
			if (!testDA.existDriver(username)) {
			driver = testDA.createDriver(username,pass);
			driverCreated = true;
			} else driver = sut.getDriver(username);
			testDA.close();*/
			sut.open();

			// Run test
			boolean u=sut.gauzatuEragiketa(username, amount, deposit);

			// Check function success and correct result
			assertFalse(u);

		}catch(NullPointerException e){
			// Uncaught exception
			fail();
		}catch(Exception e) {
			e.printStackTrace();
			fail();
		} finally{
			// Cleanup
			/*testDA.open();
			if (driverCreated)
			testDA.removeDriver(username);
			testDA.close();*/
			sut.close();
		}
	}
```

```
@Test
public void negativeTest9() {
      String username = "testuser";
      String pass ="a";
      Double amount = null;
      boolean deposit = true;
      boolean driverCreated = false;

      try {

            // Add testuser to database and save User object as "driver"
            testDA.open();
            if (!testDA.existDriver(username)) {
            driver = testDA.createDriver(username,pass);
            driverCreated = true;
            } else driver = sut.getDriver(username);
            testDA.open();

            sut.open();

            // Run test
            @SuppressWarnings("null")
            boolean u=sut.gauzatuEragiketa(username, amount, deposit);

            // Should catch null pointer if it happens and return false
            assertFalse(u);

      }catch(NullPointerException e) {
            // Uncaught exception
            fail();
      }catch(Exception e) {
            e.printStackTrace();
            fail();
      } finally{
            // Cleanup
            testDA.open();
            if (driverCreated)
            testDA.removeDriver(username);
```

```
			testDA.close();
			sut.close();
		}
	}

	@Test
	public void negativeTest10() {
		String username = "testuser";
		String pass ="a";
		double amount = 10;
		Boolean deposit = null;
		boolean driverCreated = false;

		try {

			// Add testuser to database and save User object as "driver"
			testDA.open();
			if (!testDA.existDriver(username)) {
			driver = testDA.createDriver(username,pass);
			driverCreated = true;
			} else driver = sut.getDriver(username);
			testDA.open();

			sut.open();

			// Run test deposit = null
			@SuppressWarnings("null")
			boolean u=sut.gauzatuEragiketa(username, amount, deposit);

			// Correct if function
			assertFalse(u);

		}catch(NullPointerException e) {
			// Uncaught Exception is thrown, program failed. No checks are
done for this error.
			fail();
		}catch(Exception e) {
			e.printStackTrace();
```

```
                fail();
            } finally{
                // Cleanup
                testDA.open();
                if (driverCreated)
                testDA.removeDriver(username);
                testDA.close();
                sut.close();
            }
        }


    }
```

GauzatuEragiketaMockBlackTest

```
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;
import static org.junit.Assert.fail;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;
import javax.persistence.TypedQuery;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.mockito.Mock;
import org.mockito.MockedStatic;
import org.mockito.Mockito;
import org.mockito.MockitoAnnotations;

import dataAccess.DataAccess;
import domain.User;

public class GauzatuEragiketaMockBlackTest {
static DataAccess sut;

     protected MockedStatic<Persistence> persistenceMock;

     @Mock
     protected  EntityManagerFactory entityManagerFactory;
     @Mock
     protected  EntityManager db;
     @Mock
     protected  EntityTransaction  et;

     @Mock
     TypedQuery<User> typedQuery;
```

```
      @Before
      public  void init() {
      MockitoAnnotations.openMocks(this);
      persistenceMock = Mockito.mockStatic(Persistence.class);
      persistenceMock.when(() ->
Persistence.createEntityManagerFactory(Mockito.any()))
      .thenReturn(entityManagerFactory);

      Mockito.doReturn(db).when(entityManagerFactory).createEntityManager();
      Mockito.doReturn(et).when(db).getTransaction();
      sut=new DataAccess(db);
      }
      @After
      public  void tearDown() {
      persistenceMock.close();
      }

      @SuppressWarnings({ "unchecked" })
      @Test
      public void positiveTest1() {
      String username = "testuser";
      double amount = 10;
      boolean deposit = true;

      String mota = "admin";
      String pass = "a";

      try {

            User user = new User(username, pass, mota);
            user.setMoney(10);

            Mockito.when(db.createQuery(Mockito.anyString(),
Mockito.any(Class.class))).thenReturn(typedQuery);
            Mockito.when(typedQuery.getSingleResult()).thenReturn(user);

            double expected = user.getMoney()+amount;
```

```
			boolean u=sut.gauzatuEragiketa(username, amount, deposit);

			double current = user.getMoney();

			assertTrue(u);
			assertEquals(expected, current, 0.001);

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

		}

		@SuppressWarnings({ "unchecked" })
		@Test
		public void positiveTest2() {
		String username ="testuser";
		double amount = 5;
		boolean deposit = false;

		String pass ="a";
		String mota ="admin";


		try {
			User user = new User(username, pass, mota);
			user.setMoney(10);

			double expected = user.getMoney() - amount;

			Mockito.when(db.createQuery(Mockito.anyString(),
Mockito.any(Class.class))).thenReturn(typedQuery);
			Mockito.when(typedQuery.getSingleResult()).thenReturn(user);

			boolean u=sut.gauzatuEragiketa(username, amount, deposit);
			double current = sut.getUser(username).getMoney();

			assertTrue(u);
			assertEquals(expected , current, 0.001);

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

		@SuppressWarnings({ "unchecked" })
		@Test
		public void positiveTest3() {
		String username = "testuser";
		String pass ="a";
		String mota ="admin";
		double amount = 15;
		boolean deposit = false;

		try {

			User user = new User(username, pass, mota);
			user.setMoney(10);

			Mockito.when(db.createQuery(Mockito.anyString(),
Mockito.any(Class.class))).thenReturn(typedQuery);
			Mockito.when(typedQuery.getSingleResult()).thenReturn(user);

			boolean u=sut.gauzatuEragiketa(username, amount, deposit);

			assertTrue(u);
			// Expected 0 because amount to deduct is bigger than money set, and
wallet cannot be negative
			assertEquals(0 , user.getMoney(), 0.001);

		}catch(Exception e) {
			e.printStackTrace();
```

```
            fail();
      }

      }

      @SuppressWarnings({ "unchecked" })
      @Test
      public void positiveTest4() {
      String username = "testuser";
      String pass ="a";
      String mota ="admin";
      double amount = 10;
      boolean deposit = false;

      try {

            User user = new User(username, pass, mota);
            user.setMoney(10);

            Mockito.when(db.createQuery(Mockito.anyString(),
Mockito.any(Class.class))).thenReturn(typedQuery);
            Mockito.when(typedQuery.getSingleResult()).thenReturn(user);

            boolean u=sut.gauzatuEragiketa(username, amount, deposit);

            assertTrue(u);
            // Expected 0 because amount to deduct is the same as amount in wallet,
result should be 0
            assertEquals(0 , user.getMoney(), 0.001);

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

      }

      @SuppressWarnings({ "unchecked" })
```

```
	@Test
	public void negtiveTest1() {
	String username = "testuser";
	String pass ="a";
	String mota ="admin";
	double amount = 10;
	boolean deposit = false;

	try {

		User user = new User(username, pass, mota);
		user.setMoney(20);

		Mockito.when(db.createQuery(Mockito.anyString(),
Mockito.any(Class.class))).thenReturn(typedQuery);
		Mockito.when(typedQuery.getSingleResult()).thenReturn(null);

		boolean u=sut.gauzatuEragiketa(username, amount, deposit);

		assertFalse(u);

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

	}

	@SuppressWarnings({ "unchecked" })
	@Test
	public void negtiveTest2() {
	String username = "12345";
	String pass ="a";
	String mota ="admin";
	double amount = 10;
	boolean deposit = false;

	try {
```

```
            User user = new User(username, pass, mota);
            user.setMoney(20);

            Mockito.when(db.createQuery(Mockito.anyString(),
Mockito.any(Class.class))).thenReturn(typedQuery);
            Mockito.when(typedQuery.getSingleResult()).thenReturn(user);

            boolean u=sut.gauzatuEragiketa(username, amount, deposit);

            // Numeric string usernames being accepted might be unintended
            assertFalse(u);

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

      }

      @SuppressWarnings({ "unchecked" })
      @Test
      public void negtiveTest3() {
      String username = "";
      String pass ="a";
      String mota ="admin";
      double amount = 10;
      boolean deposit = false;

      try {

            User user = new User(username, pass, mota);
            user.setMoney(20);

            Mockito.when(db.createQuery(Mockito.anyString(),
Mockito.any(Class.class))).thenReturn(typedQuery);
            Mockito.when(typedQuery.getSingleResult()).thenReturn(user);
            boolean u=sut.gauzatuEragiketa(username, amount, deposit);

            // Empty username should not be accepted
            assertFalse(u);

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

      }


      @SuppressWarnings({ "unchecked" })
      @Test
      public void negtiveTest5() {
      String username = "testuser";
      String pass ="a";
      String mota ="admin";
      double amount = -10;
      boolean deposit = false;

      try {

            User user = new User(username, pass, mota);
            user.setMoney(20);

            Mockito.when(db.createQuery(Mockito.anyString(),
Mockito.any(Class.class))).thenReturn(typedQuery);
            Mockito.when(typedQuery.getSingleResult()).thenReturn(user);

            boolean u=sut.gauzatuEragiketa(username, amount, deposit);

            // Check if operation is done with non-valid numbers
            // If operation goes through, baseline amount should change and lead to
failure
            assertEquals(10, sut.getActualMoney(username), 0.001);
```

```
		// Negative numbers should not be accepted.
		// It defeats the point of the deposit parameter.
		assertFalse(u);

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

	}

	@SuppressWarnings({ "unchecked" })
	@Test
	public void negtiveTest6() {
	String username = "testuser";
	String pass ="a";
	String mota ="admin";
	double amount = 10;
	boolean deposit = false;

	try {

		User user = new User(username, pass, mota);
		user.setMoney(20);

		Mockito.when(db.createQuery(Mockito.anyString(), 
Mockito.any(Class.class))).thenReturn(typedQuery);
		Mockito.when(typedQuery.getSingleResult()).thenReturn(user);

		boolean u=sut.gauzatuEragiketa(username, amount, deposit);

		// Operation with amount 0 is redundant and should not be accepted.
		assertFalse(u);

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


	@SuppressWarnings({ "unchecked" })
	@Test
	public void negtiveTest8() {
	String username = null;
	String pass ="a";
	String mota ="admin";
	double amount = 10;
	boolean deposit = false;

	try {

		User user = new User(username, pass, mota);
		user.setMoney(20);

		Mockito.when(db.createQuery(Mockito.anyString(),
Mockito.any(Class.class))).thenReturn(typedQuery);
		Mockito.when(typedQuery.getSingleResult()).thenReturn(user);

		boolean u=sut.gauzatuEragiketa(username, amount, deposit);

		// Possible NullPointerException caught and/or handled
		assertFalse(u);
	}catch(NullPointerException e) {
		// Uncaught Exception
		fail();
	}catch(Exception e) {
		e.printStackTrace();
		fail();
	}

	}

	@SuppressWarnings({ "unchecked" })
	@Test
	public void negtiveTest9() {
	String username = "testuser";
	String pass ="a";
	String mota ="admin";
	Double amount = null;
	boolean deposit = false;

	try {

		User user = new User(username, pass, mota);

		Mockito.when(db.createQuery(Mockito.anyString(),
Mockito.any(Class.class))).thenReturn(typedQuery);
		Mockito.when(typedQuery.getSingleResult()).thenReturn(user);

		@SuppressWarnings("null")
		boolean u=sut.gauzatuEragiketa(username, amount, deposit);

		// Possible NullPointerException caught and/or handled
		assertFalse(u);

	}catch (NullPointerException e){
		// Uncaught Exception
		fail();
	}catch(Exception e) {
		e.printStackTrace();
		fail();
	}

	}

	@SuppressWarnings({ "unchecked" })
	@Test
	public void negtiveTest10() {
	String username = "testuser";
	String pass ="a";
	String mota ="admin";
	double amount = 10;
```

```
      Boolean deposit = null;

      try {

            User user = new User(username, pass, mota);
            user.setMoney(20);

            Mockito.when(db.createQuery(Mockito.anyString(),
Mockito.any(Class.class))).thenReturn(typedQuery);
            Mockito.when(typedQuery.getSingleResult()).thenReturn(user);

            @SuppressWarnings("null")
            boolean u=sut.gauzatuEragiketa(username, amount, deposit);

            // Possible NullPointerException caught and/or handled
            assertFalse(u);
      }catch (NullPointerException e){
            // Uncaught Exception
            fail();
      }catch(Exception e) {
            e.printStackTrace();
            fail();
      }

      }

}
```

GauzatuEragiketaBDWhiteTest

```
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;
import static org.junit.Assert.fail;

import org.junit.Test;

import dataAccess.DataAccess;
import domain.Driver;
import domain.User;
import testOperations.TestDataAccess;

public class GauzatuEragiketaBDWhiteTest {

	//sut:system under test
	static DataAccess sut=new DataAccess();

	//additional operations needed to execute the test
	static TestDataAccess testDA=new TestDataAccess();

	@SuppressWarnings("unused")
	private Driver driver;

	@SuppressWarnings({ "unchecked" })
	@Test
	public void test1() {

	String username = "testuser";//testuser is not in DB

	double amount = 10;
	boolean deposit = true;


	try {
		sut.open();
```

```
            boolean u=sut.gauzatuEragiketa(username, amount, deposit);
            // Looking for a non-existent user is the only way I can think of but
this throws an exception rather than returning null.
            // The test takes an unintended path so it's technically a failure.
            assertFalse(u);


      }catch(Exception e) {
            e.printStackTrace();
            fail();
      } finally{
            sut.close();
      }

      }

      @SuppressWarnings({ "unchecked" })
      @Test
      public void test2() {
      String username ="testuser";
      String pass ="a";
      double amount = 10;
      boolean deposit = true;
      boolean driverCreated = false;


      try {
            // Add testuser to database and save User object as "driver"
            testDA.open();
            if (!testDA.existDriver(username)) {
                  driver = testDA.createDriver(username,pass);
                  driverCreated = true;
            } else driver = sut.getDriver(username);
            testDA.open();

            // Get expected money, current + amount to add
            double expected = driver.getMoney() + amount;
```

```
		sut.open();
		// Run test
		boolean u=sut.gauzatuEragiketa(username, amount, deposit);

		// Get new money amount
		double current= sut.getDriver(username).getMoney();

		// Check function success and correct result
		assertTrue(u);
		assertEquals(expected , current, 0.001);

	}catch(Exception e) {
		e.printStackTrace();
		fail();
	} finally{
		// Cleanup
		testDA.open();
		if (driverCreated)
			testDA.removeDriver(username);
		testDA.close();
		sut.close();
	}

	}

	@SuppressWarnings({ "unchecked" })
	@Test
	public void test3() {
	String username = "testuser";
	String pass ="a";
	double amount;
	boolean deposit = false;
	boolean driverCreated = false;

	try {

		// Add testuser to database and save User object as "driver"
		testDA.open();
		if (!testDA.existDriver(username)) {
			driver = testDA.createDriver(username,pass);
			driverCreated = true;
		} else driver = sut.getDriver(username);
		testDA.open();

		// Get expected money, which should be 0
		double expected = 0;

		// Set amount to be bigger than current money for negative result
		amount = driver.getMoney() + 10;

		sut.open();
		// Run test
		boolean u=sut.gauzatuEragiketa(username, amount, deposit);

		// Get new money amount
		double current= sut.getDriver(username).getMoney();

		// Check function success and correct result
		assertTrue(u);
		assertEquals(expected , current, 0.001);

	}catch(Exception e) {
		e.printStackTrace();
		fail();
	} finally{
		// Cleanup
		testDA.open();
		if (driverCreated)
			testDA.removeDriver(username);
		testDA.close();
		sut.close();
	}

	}

	@Test
```

```
public void test4() {

String username = "testuser";
String pass ="a";
double amount = 10;
boolean deposit = false;
boolean driverCreated = false;

try {

      // Add testuser to database and save User object as "driver"
      testDA.open();
      if (!testDA.existDriver(username)) {
            driver = testDA.createDriver(username,pass);
            driverCreated = true;
      } else driver = sut.getDriver(username);
      testDA.open();

      sut.open();

      // Add money to driver for positive result
      sut.gauzatuEragiketa(username, 100, true);

      // Set expected result, which should be current - amount
      double expected = 100 - amount;

      // Run test
      boolean u=sut.gauzatuEragiketa(username, amount, deposit);

      // Get new money amount
      double current= sut.getDriver(username).getMoney();

      // Check function success and correct result
      assertTrue(u);
      assertEquals(expected , current, 0.001);

}catch(Exception e) {
      e.printStackTrace();
      fail();
} finally{
      // Cleanup
      testDA.open();
      if (driverCreated)
            testDA.removeDriver(username);
      testDA.close();
      sut.close();
}

}

@SuppressWarnings({ "unchecked" })
@Test
public void test5() {
String username = null;
double amount = 10;
boolean deposit = false;

try {

      sut.open();
      // Run test
      boolean u=sut.gauzatuEragiketa(username, amount, deposit);
      // null username makes db.getUser(String) throw a NoResultException
      // gauzatuEragiketa catches the exception properly and returns false

      assertFalse(u);

}catch(Exception e) {
      e.printStackTrace();
      fail();
} finally{
      // Cleanup
      sut.close();
}

}
```

```
}
```

GauzatuEragiketaMockWhiteTest

```
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;
import static org.junit.Assert.fail;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;
import javax.persistence.TypedQuery;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.mockito.Mock;
import org.mockito.MockedStatic;
import org.mockito.Mockito;
import org.mockito.MockitoAnnotations;

import dataAccess.DataAccess;
import domain.User;

public class GauzatuEragiketaMockWhiteTest {
static DataAccess sut;

     protected MockedStatic<Persistence> persistenceMock;

     @Mock
     protected  EntityManagerFactory entityManagerFactory;
     @Mock
     protected  EntityManager db;
     @Mock
     protected  EntityTransaction  et;

     @Mock
     TypedQuery<User> typedQuery;

     @Before
     public  void init() {
     MockitoAnnotations.openMocks(this);
     persistenceMock = Mockito.mockStatic(Persistence.class);
     persistenceMock.when(() ->
Persistence.createEntityManagerFactory(Mockito.any()))
     .thenReturn(entityManagerFactory);

     Mockito.doReturn(db).when(entityManagerFactory).createEntityManager();
     Mockito.doReturn(et).when(db).getTransaction();
     sut=new DataAccess(db);
     }
     @After
     public  void tearDown() {
     persistenceMock.close();
     }

     @SuppressWarnings({ "unchecked" })
     @Test
     public void test1() {
     String username="testuser";
     double amount= 10;
     boolean deposit= true;


     try {
           Mockito.when(db.createQuery(Mockito.anyString(),
Mockito.any(Class.class))).thenReturn(typedQuery);
           Mockito.when(typedQuery.getSingleResult()).thenReturn(null);

           boolean u=sut.gauzatuEragiketa(username, amount, deposit);
```

```
            assertFalse(u);


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

      }

      @SuppressWarnings({ "unchecked" })
      @Test
      public void test2() {
      String username ="testuser";
      String pass ="a";
      String mota ="admin";
      double amount = 10;
      boolean deposit = true;


      try {
            User user = new User(username, pass, mota);
            Double money = user.getMoney();

            Mockito.when(db.createQuery(Mockito.anyString(), 
Mockito.any(Class.class))).thenReturn(typedQuery);
            Mockito.when(typedQuery.getSingleResult()).thenReturn(user);

            boolean u=sut.gauzatuEragiketa(username, amount, deposit);

            assertTrue(u);
            assertEquals(money+amount , user.getMoney(), 0.001);

      }catch(Exception e) {
            e.printStackTrace();
            fail();
      }
```

```
	}

	@SuppressWarnings({ "unchecked" })
	@Test
	public void test3() {
	String username = "testuser";
	String pass ="a";
	String mota ="admin";
	double amount = 10;
	boolean deposit = false;

	try {

		User user = new User(username, pass, mota);
		user.setMoney(5);

		Mockito.when(db.createQuery(Mockito.anyString(),
Mockito.any(Class.class))).thenReturn(typedQuery);
		Mockito.when(typedQuery.getSingleResult()).thenReturn(user);

		boolean u=sut.gauzatuEragiketa(username, amount, deposit);

		assertTrue(u);
		assertEquals(0 , user.getMoney(), 0.001);

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

	}

	@SuppressWarnings({ "unchecked" })
	@Test
	public void test4() {
	String username = "testuser";
	String pass ="a";
	String mota ="admin";
	double amount = 10;
	boolean deposit = false;

	try {

		User user = new User(username, pass, mota);
		user.setMoney(20);

		Mockito.when(db.createQuery(Mockito.anyString(), 
Mockito.any(Class.class))).thenReturn(typedQuery);
		Mockito.when(typedQuery.getSingleResult()).thenReturn(user);

		boolean u=sut.gauzatuEragiketa(username, amount, deposit);

		assertTrue(u);
		assertEquals(10 , user.getMoney(), 0.001);

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

	}

	@SuppressWarnings({ "unchecked" })
	@Test
	public void test5() {
	String username = "testuser";
	String pass ="a";
	String mota ="admin";
	double amount = 10;
	boolean deposit = false;

	try {

		User user = new User(username, pass, mota);
		user.setMoney(20);
```

```
			Mockito.when(db.createQuery(Mockito.anyString(), 
Mockito.any(Class.class))).thenReturn(typedQuery);
			Mockito.when(typedQuery.getSingleResult()).thenReturn(user);
			Mockito.doThrow(new RuntimeException("Commit 
failed")).when(et).commit();

			boolean u=sut.gauzatuEragiketa(username, amount, deposit);

			assertFalse(u);

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

		}

}
```