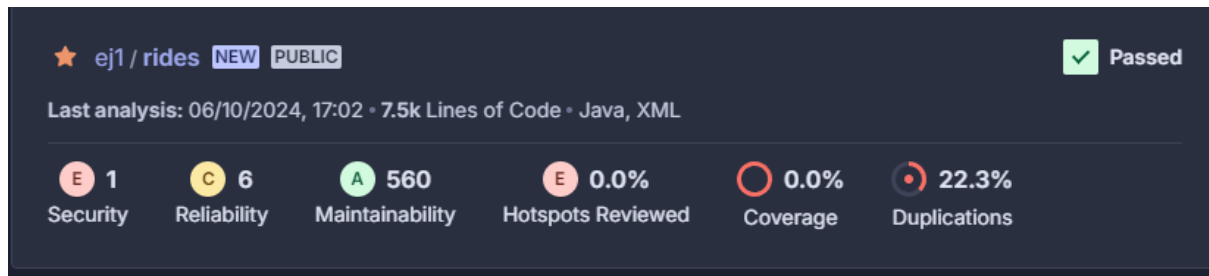


Proyecto verificación del Software

(Mikel Calleja)

Corrección de issues con sonarLint

Estado inicial en Sonar Cloud:



Issues corregidos:

Del tipo duplicación, en el método AdminGUI se hacía uso varias veces del String “Etiqueta” para solucionar cree una variable de tipo String.

```
public AdminGUI(String username) {  
  
    String e = "Etiquetas";  
    AdminGUI.setBusinessLogic(LoginGUI.getBusinessLogic());  
  
    this.setTitle(ResourceBundle.getBundle(e).getString("AdminGUI.Admin"));  
    this.setSize(495, 290);  
  
    JLabelSelectOption = new JLabel(ResourceBundle.getBundle(e).getString("AdminGUI.Admin"));  
    JLabelSelectOption.setFont(new Font("Tahoma", Font.BOLD, 13));  
    JLabelSelectOption.setForeground(Color.BLACK);  
    JLabelSelectOption.setHorizontalAlignment(SwingConstants.CENTER);  
  
    JButtonDeskontu = new JButton();  
    JButtonDeskontu.addActionListener(actionEvent -> {  
        JFrame a = new DeskontuaGUI(username);  
        a.setVisible(true);  
    });  
  
    JButtonDeskontu.setText(ResourceBundle.getBundle(e).getString("AdminGUI.Deskontua"));  
  
    JButtonkude = new JButton();  
    JButtonkude.addActionListener(actionEvent -> {  
        JFrame a = new DeskontuKudeatuGUI(username);  
        a.setVisible(true);  
    });  
  
    JButtonkude.setText(ResourceBundle.getBundle(e).getString("AdminGUI.Kudea"));  
  
    JButtonEzabatu = new JButton();  
    JButtonEzabatu.setText(ResourceBundle.getBundle(e).getString("AdminGUI.Ezab"));  
    JButtonEzabatu.addActionListener(actionEvent -> {  
        JFrame a = new EzabatuGUI();  
        a.setVisible(true);  
    });  
}
```

En otra clase para solucionar cree una constante arriba de esta manera:
private static final String ETIQUETA = "etiqueta"; si lo hacia usadno simplemente como et
no cumpla match the regular expression `^[A-Z][A-Z0-9]*(_[A-Z0-9]+)*$`.

```

public class AlertaSortuGUI extends JFrame {
    private static final long serialVersionUID = 1L;

    private JTextField fieldOrigin = new JTextField();
    private JTextField fieldDestination = new JTextField();
    private static final String ETIQUETA = "etiqueta";

    private JLabel jLabelOrigin = new JLabel(
        ResourceBundle.getBundle(ETIQUETA).getString("CreateRideGUI.LeavingFrom"));
    private JLabel jLabelDestination = new JLabel(
        ResourceBundle.getBundle(ETIQUETA).getString("CreateRideGUI.GoingTo"));
    private JLabel jLabelRideDate = new JLabel(ResourceBundle.getBundle(ETIQUETA).getString("CreateRideGUI.RideDate"));

    private JCalendar jCalendar = new JCalendar();
    private Calendar calendarAct = null;
    private Calendar calendarAnt = null;

    @SuppressWarnings("unused")
    private List<Date> datesWithEventsCurrentMonth;

    private JScrollPane scrollPaneEvents = new JScrollPane();

    private JButton jButtonCreate = new JButton(ResourceBundle.getBundle(ETIQUETA).getString("AlertGUI.AddAlert"));
    private JButton jButtonClose = new JButton(ResourceBundle.getBundle(ETIQUETA).getString("Close"));
    private JLabel jLabelMsg = new JLabel();
    private JLabel jLabelError = new JLabel();

    private Traveler traveler;

    private static BLFacade appFacadeInterface;

    public static BLFacade getBusinessLogic() {
        return appFacadeInterface;
    }

    public static void setBusinessLogic(BLFacade afi) {
        appFacadeInterface = afi;
    }

    public AlertaSortuGUI(String username) {

```

Por otro lado en la clase Admin2 había un par de issues relacionados con replace this if-then-else statement by a single return statement. Además de incluir a “hashCode()”

antes →

```

public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Admin2 other = (Admin2) obj;
    if (username != other.username)
        return false;
    return true;
}

```

Después →

```

@Override
public int hashCode() {

```

```

        return Objects.hash(username);
    }
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null || getClass() != obj.getClass())
            return false;
        Admin2 other = (Admin2) obj;
        return Objects.equals(username, other.username);
    }

```

También corregí varios issues del tipo rename que cumplan `^[A-Z][A-Z0-9]*(_[A-Z0-9]+)*$`. Como por ejemplo los siguientes métodos:

- jButtonClose_actionPerformed → jButtonCloseActionPerformed
- closeButton_actionPerformed → closeButtonActionPerformed
- jButtonClose_actionPerformed → jButtonCloseActionPerformed
- field_Errors → fieldErrors

Otro de los issues era make this anonymous inner class a lambda, una de las modificaciones que hice fue esta:

Antes →

```

jButtonDeskontu.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        JFrame a = new DeskontuaGUI(username);
        a.setVisible(true);}
});

```

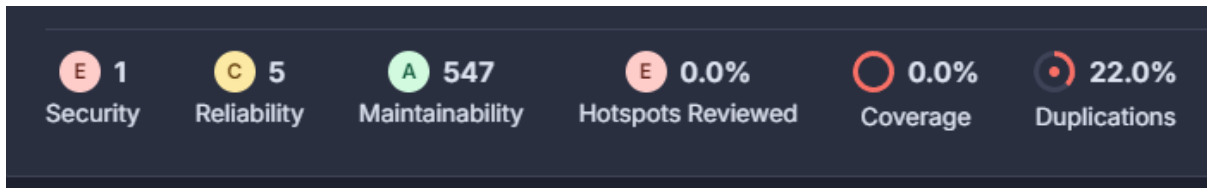
Después →

```

jButtonDeskontu.addActionListener(new ActionListener() { public void
    actionPerformed(ActionEvent e) { JFrame a = new DeskontuaGUI(username);
        a.setVisible(true); } });

```

Después de solucionar issues:



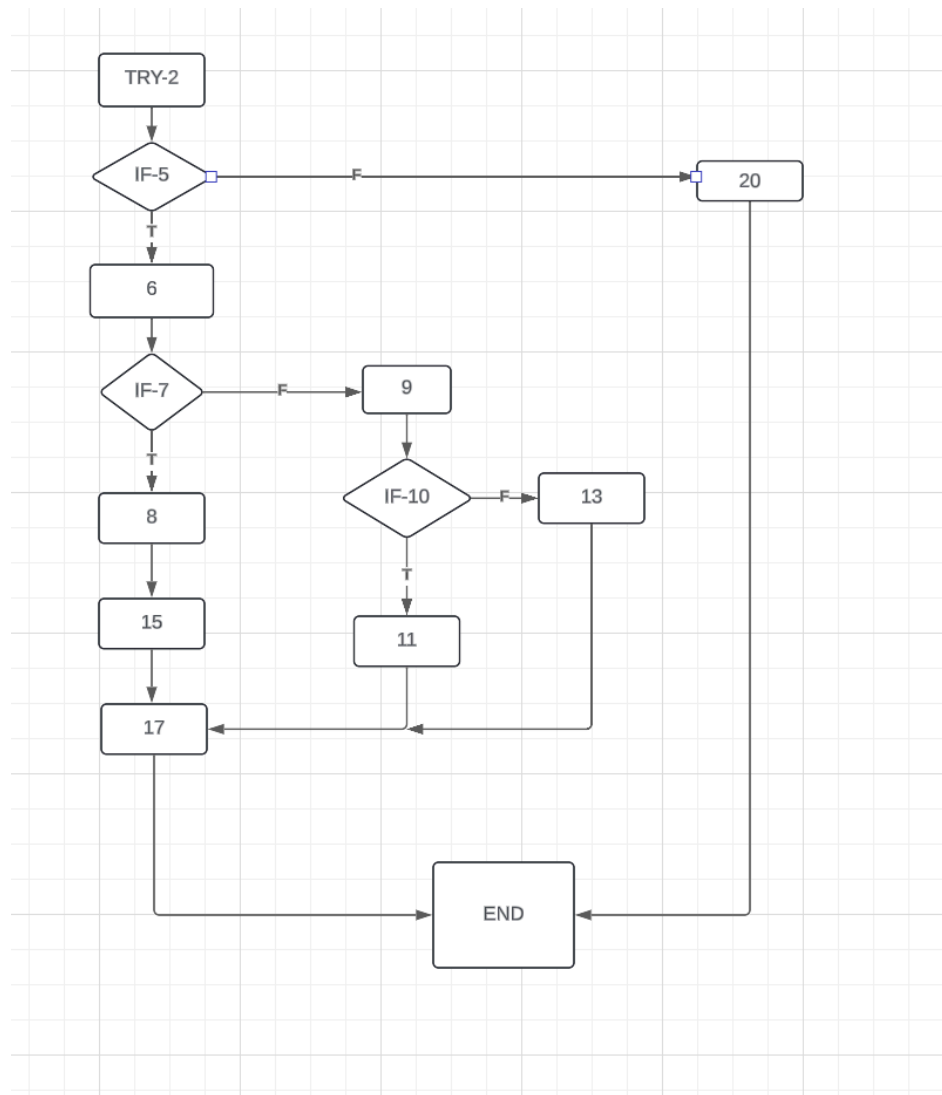
Link de sonar → https://sonarcloud.io/project/issues?issueStatuses=OPEN%2CCONFIRMED&id=ej1_rides

Método escogido gauzatuEragiketa (Mikel Calleja):

Link GitHub → <https://github.com/SikeMike/IS2-Rides24.git>

```
public boolean gauzatuEragiketa(String username, double amount, boolean deposit)
{
    try {
        db.getTransaction().begin();
        User user = getUser(username);
        System.out.println(user);
        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;
    }
}
```

Grafo de flujo de control:



La complejidad de este método es de $4+1$; $V(G)=5$

Caja blanca:

Casos de prueba:

		Contexto de prueba		Resultado esperado	
#	Condición	Estado BD	Entrada	Estado BD	Salida
1	IF-5(F)	u ∉ DB	u="pepe" 150 true	*	NoResultException
2	IF-5(T), IF-7(T)	U ∈ DB	"Juan" 150 true	Juan money amount +150	TRUE
3	IF-5(T), IF-7(T), IF-10(T)	U ∈ DB	"Juan" 150 False	Juan money 0	TRUE
4	IF-5(T), IF-7(T), IF-10(F)	U ∈ DB	"Juan" 50 False	Juan money amount -50	TRUE

Implementaciones de Caja Blanca

GauzatuEragiketaBDWhiteTest.java y GauzatuEragiketaMockW.java

GauzatuEragiketaMockW.java →
(algunos test no están incluidos que ocupaban mucho)

```
@Test
public void test1() {
    try {
        when(db.find(User.class, "username")).thenReturn(null);
        Mockito.when(db.createQuery(Mockito.anyString(),
Mockito.any(Class.class))).thenReturn(typedQueryDouble);
        Mockito.when(typedQueryDouble.getSingleResult()).thenReturn(null);

        boolean result = sut.gauzatuEragiketa("username", 100.0, true);

        // Verificar
        assertFalse(result);

        verify(et).begin();
        verify(et).commit();
        verify(et, never()).rollback();
    } catch (NullPointerException e) {
        System.out.println("Error: NullPointerException.");
        e.printStackTrace();
        fail("NullPointerException fue lanzada");
    }
}

@Test
public void test2() {
    try {
        Mockito.when(db.createQuery(Mockito.anyString(),
Mockito.any(Class.class))).thenReturn(typedQueryDouble);
        Mockito.when(typedQueryDouble.getSingleResult()).thenReturn(null);

        boolean result = sut.gauzatuEragiketa(null, 100.0, true);

        // Verificar
        assertFalse(result);

        verify(et).begin();
        verify(et).commit();
        verify(et, never()).rollback();
    } catch (NullPointerException e) {
```

```

        System.out.println("Error: NullPointerException.");
        e.printStackTrace();
        fail("NullPointerException fue lanzada");
    }
}

@Test
public void test3() {

    User user = new User("Juan", "1234a", "a");

    try {

        when(db.find(User.class, "Juan")).thenReturn(user);
        Mockito.when(db.createQuery(Mockito.anyString(), Mockito.any(Class.class))).thenReturn(typedQueryUser);
        Mockito.when(typedQueryUser.getSingleResult()).thenReturn(user);

        boolean result = sut.gauzatuEragiketa("Juan", 100.0, true);

        // Verificar
        assertTrue(result);

        verify(et).begin();
        verify(et).commit();
        verify(et, never()).rollback();

    } catch (NullPointerException e) {
        System.out.println("Error: NullPointerException.");
        e.printStackTrace();
        fail("NullPointerException fue lanzada");
    }
}

@Test
public void testDepositUserFound() {
    try {

        User user = new User("Juan", "1234a", "a");
        user.setMoney(100.0);

        TypedQuery<User> typedQueryUser = Mockito.mock(TypedQuery.class);

        Mockito.when(db.createQuery(Mockito.anyString(), Mockito.any(Class.class))).thenReturn(typedQueryUser);

        Mockito.when(typedQueryUser.getSingleResult()).thenReturn(user);

        Mockito.when(typedQueryUser.setParameter(Mockito.anyString(),
Mockito.anyString())).thenReturn(typedQueryUser);

        boolean result = sut.gauzatuEragiketa("username", 50.0, true);

        assertTrue(result);
        assertEquals(150.0, user.getMoney(), 0.01);

        verify(db).merge(user);
        verify(et).begin();
        verify(et).commit();
        verify(et, never()).rollback();
    } catch (NullPointerException e) {
        fail("NullPointerException fue lanzada.");
    }
}

@Test
public void testDepositUserSufficientFunds() {
    try {

        User user = new User("Juan", "1234a", "a");
        user.setMoney(100.0);

        TypedQuery<User> typedQueryUser = Mockito.mock(TypedQuery.class);
        Mockito.when(db.createQuery(Mockito.anyString(), Mockito.any(Class.class))).thenReturn(typedQueryUser);
        Mockito.when(typedQueryUser.getSingleResult()).thenReturn(user);
    }
}

```

```

        Mockito.when(typedQueryUser.setParameter(Mockito.anyString(),
Mockito.any()))
.thenReturn(typedQueryUser);

        boolean result = sut.gauzatuEragiketa("username", 50.0, true);

        assertTrue(result);
        assertEquals(150.0, user.getMoney(), 0.01);

        verify(db).merge(user);
        verify(et).begin();
        verify(et).commit();
        verify(et, never()).rollback();
    } catch (NullPointerException e) {
        fail("NullPointerException fue lanzada.");
    }
}

```

GauzatuEragiketaBDWhiteTest.java →

@Test

```

// Debería devolver verdadero, porque "Urtzi2" existe
public void test1() {
    boolean a = false;
    try {
        // define parameters
        String driverUsername = "Urtzi2"; // Un driver que existe
        // invoke System Under Test (sut)
        a = testDA.gauzatuEragiketa(driverUsername, 150, true);
        // imprimir el resultado y el estado del usuario
        System.out.println("Resultado de gauzatuEragiketa: " + a);
        User user = testDA.getUser(driverUsername);
        System.out.println("Dinero de " + driverUsername + ": " + user.getMoney()); // Ver dinero
        assertTrue(a); // Debería ser verdadero
    } catch (Exception e) {
        fail();
    }
    System.out.println("Test 1 fin");
}

@Test
public void test2() {
    try {
        boolean result = testDA.gauzatuEragiketa("Urtzi2", 50, false); // Retirar 50
        assertTrue(result);
        User user = testDA.getUser("Urtzi2");
        assertEquals(160, user.getMoney(), 0);
    } catch (Exception e) {
        fail(); // Si hay una excepción, la prueba falla
    }
    System.out.println("Test 2 fin");
}

@Test
public void test3() {
    try {
        boolean result = testDA.gauzatuEragiketa("Urtzi2", 20, false); // Retirar 20
        assertTrue(result);
        User user = testDA.getUser("Urtzi2");
        assertEquals(195, user.getMoney(), 0); // Deberia dar 195
    } catch (Exception e) {
        fail();
    }
    System.out.println("Test 3 fin");
}

@Test
public void test4() {
    try {
        boolean result = testDA.gauzatuEragiketa("NoExistente", 100, true); // Usuario no existe
        assertFalse(result); // Debe devolver false
    }
}

```



```

    } catch (Exception e) {
        fail();
    }
    System.out.println("Test 4 fin");
}
@Test
public void test5() {
    boolean result = testDA.gauzatuEragiketa("Urtzi2", 50, true); // Depositar 50
    assertTrue(result);
    User user = testDA.getUser("Urtzi2");
    assertEquals(265, user.getMoney(), 0); // Verifica que el saldo sea 65
    System.out.println("Test 5 fin");
}

@Test
public void test6() {
    boolean result = testDA.gauzatuEragiketa("Urtzi2", -50, false); // Depositar -50
    assertTrue(result);
    User user = testDA.getUser("Urtzi2");
    System.out.println("El saldo no deberia dar 265 si no 165"+user.getMoney());
    assertEquals(165, user.getMoney(), 0); // Verifica que el saldo sea 165 y no sea 265
    System.out.println("Test 6 fin");
}
}

```

Implementaciones de Caja Negra

Casos de prueba:

Condición entrada	Clases de equivalencia validas	Clases de equivalencia inválidas
Valor de username	username = "user1" (1)	username == null (11)
	username = "user2" (2)	username = "" (12)
	username = "validUser" (3)	username = "nonExistentUser" (13)
Valor de amount	amount > 0 (4)	amount < 0 (14)
	amount = 0 (5)	amount = Double.NaN (15)
valor de deposit	deposit = true (6)	deposit = false (16)
	deposit = false (7)	

#	Entrada	Clases cubiertas	Estado BD	Salida
1	("user1", 100.0, true)	(1,2, 4, 6)	U ∈DB	TRUE
2	("user1", 50.0, false)	(1,2, 4, 7)	U ∈DB	TRUE
3	("user1", 200.0, false)	(1,2, 4, 7)	U ∈DB	TRUE
4	("PEPE", 50.0, false)	(1,2, 4, 7)	u∉DB	FALSE
5	("user1", -50.0, true)	(1,2, 14, 6)	U ∈DB	FALSE
6	("user1", 0.0, true)	(1,2, 5, 6)	U ∈DB	TRUE
7	("user1", 50.0, true)	(1,2, 4, 6)	U ∈DB	TRUE
8	("user1", 50.0, false)	(1,2, 4, 7)	U ∈DB	TRUE
9	("", 100.0, true)	(1,12, 4, 6)	u∉DB	FALSE
10	(null, 50.0, false)	(11, 4, 7)	U ∈DB	FALSE
11	("user1", Double.NaN, true)	(1,2, 15, 6)	U ∈DB	FALSE
12	("user1", 100.0, false)	(1,2, 4, 7)	U ∈DB	TRUE

GauzatuEragiketaBDBlackTest.java y GauzatuEragiketaMockB.java

GauzatuEragiketaMockB.java →

```

public class GauzatuEragiketaMockB {
    static DataAccess sut;
    protected MockedStatic<Persistence> persistenceMock;
    @Mock
    protected EntityManagerFactory entityManagerFactory;
    @Mock
    protected EntityManager db;
    @Mock
    protected EntityTransaction et;
    @Mock
    TypedQuery<Double> typedQueryDouble;
    @Mock
    TypedQuery<User> typedQueryUser;
    @SuppressWarnings("unused")
    @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();
    }
    @Test
    public void test1() {
        try {
            when(db.find(User.class, "username")).thenReturn(null);
            Mockito.when(db.createQuery(Mockito.anyString(),
Mockito.any(Class.class))).thenReturn(typedQueryDouble);

```

```

        Mockito.when(typedQueryDouble.getSingleResult()).thenReturn(null);
        boolean result = sut.gauzatuEragiketa("username", 100.0, true);
        // Verificar
        assertFalse(result);
        verify(et).begin();
        verify(et).commit();
        verify(et, never()).rollback();
    } catch (NullPointerException e) {
        System.out.println("Error: NullPointerException.");
        e.printStackTrace();
        fail("NullPointerException fue lanzada");
    }
}

@Test
public void test2() {
    User user = new User("Juan", "1234a", "a");
    user.setMoney(100.0); // El usuario tiene 100 de saldo inicialmente
    TypedQuery<User> typedQueryUser = Mockito.mock(TypedQuery.class);
    Mockito.when(db.createQuery(Mockito.anyString(),
Mockito.any(Class.class))).thenReturn(typedQueryUser);
    Mockito.when(typedQueryUser.getSingleResult()).thenReturn(user);
    // Intentar depositar un valor negativo, lo cual debería ser inválido
    boolean result = sut.gauzatuEragiketa("Juan", -50.0, true); // Depósito con -50
    // Verificar que el depósito no se realiza y que el saldo no cambia
    assertTrue(result);
    assertEquals(100.0, user.getMoney(), 0.01); // El saldo debería seguir siendo 100
    verify(db, never()).merge(user);
    verify(et).rollback();
}

@Test
public void test3() {
    User user = new User("Juan", "1234a", "a");
    user.setMoney(100.0); // El usuario tiene 100 de saldo inicialmente
    TypedQuery<User> typedQueryUser = Mockito.mock(TypedQuery.class);
    Mockito.when(db.createQuery(Mockito.anyString(),
Mockito.any(Class.class))).thenReturn(typedQueryUser);
    Mockito.when(typedQueryUser.getSingleResult()).thenReturn(user);
    // Intentar retirar un valor negativo, lo cual debería ser inválido
    boolean result = sut.gauzatuEragiketa("Juan", -50.0, false); // Retiro con -50
    // Verificar que el retiro no se realiza y que el saldo no cambia
    assertTrue(result);
    assertEquals(100.0, user.getMoney(), 0.01);
    verify(db, never()).merge(user);
    verify(et).rollback(); }}

```

GauzatuEragiketaBDBlackTest.java →

```

public class GauzatuEragiketaBDBlackTest {
    static TestDataAccess testDA;
    @Before
    public void setUp() {
        testDA = new TestDataAccess();
        testDA.open();
        testDA.removeAllDrivers();
        testDA.initializeDB();
    }
    @After

```

```

public void tearDown() {
    testDA.close();
}
@Test
public void tes1() {
    boolean result = testDA.gauzatuEragiketa("Zuri2", 150.0, true);
    assertTrue(result); // Debería ser verdadero
    User user = testDA.getUser("Zuri2");
    assertEquals(265.0, user.getMoney(), 0.01); // Verifica que el saldo sea 265
}
@Test
public void tes2() {
    boolean result = testDA.gauzatuEragiketa("Zuri2", 50.0, false);
    assertTrue(result); // Debería ser verdadero
    User user = testDA.getUser("Zuri2");
    assertEquals(65.0, user.getMoney(), 0.01); // Verifica que el saldo sea 65
}
@Test
public void tes3() {
    boolean result = testDA.gauzatuEragiketa("NoExistente", 100.0, true);
    assertFalse(result); // Debería ser falso
}
@Test
public void tes4() {
    boolean result = testDA.gauzatuEragiketa("Urtzi2", -50.0, true);
    assertTrue(result);
}
@Test
public void tes5() {
    boolean result = testDA.gauzatuEragiketa("Urtzi2", 0.0, false);
    assertTrue(result); // Debería ser verdadero
    User user = testDA.getUser("Urtzi2");
    assertEquals(215.0, user.getMoney(), 0.01); // El saldo no cambia
}
@Test
public void tes6() {
    boolean result = testDA.gauzatuEragiketa("Urtzi2", 300.0, false);
    assertTrue(result); // El saldo no debe ser negativo
    User user = testDA.getUser("Urtzi2");
    assertEquals(0.0, user.getMoney(), 0.01); // Verifica que el saldo sea 0
}

@Test
    public void test7() {
        boolean result = testDA.gauzatuEragiketa("Urtzi2", -50, true); // Depositar -50
        assertTrue(result);
        User user = testDA.getUser("Urtzi2");
        System.out.println(user.getMoney());
        assertEquals(265, user.getMoney(), 0); // Verifica que el saldo sea 65
        System.out.println("Test 7 fin");
    }

    @Test
    public void test8() {
        boolean result = testDA.gauzatuEragiketa("Urtzi2", -50, false); // Depositar -50
        assertTrue(result);
        User user = testDA.getUser("Urtzi2");
    }

```

```
        System.out.println(user.getMoney());  
        assertEquals(165, user.getMoney(), 0); // Verifica que el saldo sea  
        System.out.println("Test 8 fin");  
    }  
}
```

Defectos encontrados → En general todos han ido correctamente sin embargo el método no gestiona bien los valores negativos, así que si yo por ejemplo quería retirar -50 teniendo 100 acababa en la cuenta con 150 . Para corregirlo bastaría con hacer una comprobación inicial del parámetro amount. Junto con el de que si no encuentra el usuario no lo trata del todo bien (NoResultException). Y por último en caso de que el usuario no se encuentre, el método llama a db.getTransaction().commit() antes de retornar false.