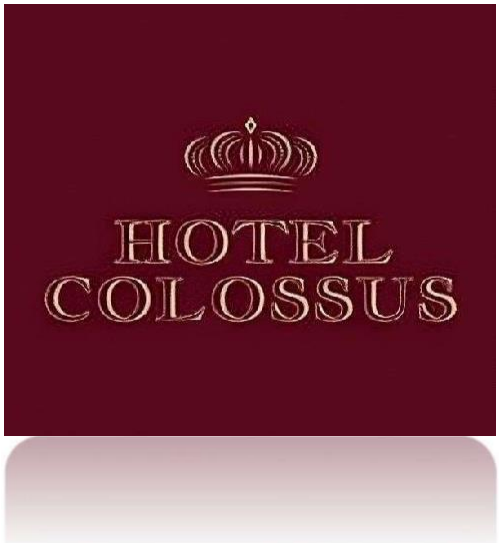
UNIVERSITÀ DEGLI STUDI
DI SALERNO



SYSTEM DESIGN DOCUMENT

TEAM COLOSSUS
Stefano Santoro – Giovanni Riccardi
-Renato Natale – Samuele Valiante

Data: 22/11/2025

Coordinatore del progetto:

Nome	Matricola
Stefano Santoro	0512120778

Partecipanti:

Nome	Matricola
Giovanni Riccardi	0512119392
Renato Natale	0512119641
Samuele Valiante	0512119125

Scritto da:	Giovanni Riccardi

Revision History

Data	Versione	Descrizione	Autore
02/02/2026	1.0	Prima versione Test Result	Giovanni Riccardi

Sommario

1. TEST RESULT BLACKBOX pg.4
2. TEST RESULT WHITEBOX pg 8

TEST RESULT BLACK BOX

CATEGORY PARTITION

Test suite: “Registrazione di una registrazione”
Test results

Test Case	Test Result
TC1	**FAIL**
TC2	**FAIL**
TC3	**FAIL**
TC4	**FAIL**
TC5	**PASS**
TC6	**PASS**
TC7	**PASS**
TC8	**PASS**
TC9	**PASS**
TC10	**PASS**
TC11	**PASS**
TC12	**PASS**
TC13	**PASS**
TC14	**PASS**
TC15	**PASS**
TC16	**PASS**
TC17	**PASS**
TC18	**PASS**
TC19	**PASS**
TC20	**PASS**

TC21	**PASS**
TC22	**PASS**
TC23	**PASS**
TC24	**PASS**

Test failures

Dopo l'esecuzione della test suite si sono presentati i seguenti failures

- TC1: riga 176
- TC1: riga 192
- TC1: riga 211
- TC1: riga 226

```
org.opentest4j.AssertionFailedError: Unexpected exception thrown: it.unisa.Server.persistent.obj
 .catalogues.InvalidInputException: Campo: metodoDiPagamento è vuoto
> <5 internal lines>
    at blackbox.RegistraPrenotazione.TestCasesRegistraPrenotazione$TestPassRegistraPrenotazione
>    .testCase1(TestCasesRegistraPrenotazione.java:176) <1 internal line>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1604)
><2 folded frames>
Caused by: it.unisa.Server.persistent.obj.catalogues.InvalidInputException Create breakpoint : Campo:
 metodoDiPagamento è vuoto
    at it.unisa.Server.persistent.obj.catalogues.CatalogueUtils.checkNull(CatalogueUtils.java:29)
    at it.unisa.Server.gestionePrenotazioni.FrontDesk.addPrenotazione(FrontDesk.java:117)
>     <6 internal lines>
><13 folded frames>
```

```
org.opentest4j.AssertionFailedError: Unexpected exception thrown: it.unisa.Server.persistent.obj
 .catalogues.InvalidInputException: Campo: metodoDiPagamento è vuoto
> <5 internal lines>
    at blackbox.RegistraPrenotazione.TestCasesRegistraPrenotazione$TestPassRegistraPrenotazione
>    .testCase2(TestCasesRegistraPrenotazione.java:192) <1 internal line>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1604)
><2 folded frames>
Caused by: it.unisa.Server.persistent.obj.catalogues.InvalidInputException Create breakpoint : Campo:
 metodoDiPagamento è vuoto
    at it.unisa.Server.persistent.obj.catalogues.CatalogueUtils.checkNull(CatalogueUtils.java:29)
    at it.unisa.Server.gestionePrenotazioni.FrontDesk.addPrenotazione(FrontDesk.java:117)
>     <6 internal lines>
><13 folded frames>
```

```
org.opentest4j.AssertionFailedError: Unexpected exception thrown: it.unisa.Server.persistent.obj
 .catalogues.InvalidInputException: Campo: metodoDiPagamento è vuoto
> <5 internal lines>
    at blackbox.RegistraPrenotazione.TestCasesRegistraPrenotazione$TestPassRegistraPrenotazione
>    .testCase3(TestCasesRegistraPrenotazione.java:211) <1 internal line>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1604)
><2 folded frames>
Caused by: it.unisa.Server.persistent.obj.catalogues.InvalidInputException Create breakpoint : Campo:
 metodoDiPagamento è vuoto
    at it.unisa.Server.persistent.obj.catalogues.CatalogueUtils.checkNull(CatalogueUtils.java:29)
    at it.unisa.Server.gestionePrenotazioni.FrontDesk.addPrenotazione(FrontDesk.java:117)
>     <6 internal lines>
><13 folded frames>
```

```
org.opentest4j.AssertionFailedError: Unexpected exception thrown: it.unisa.Server.persistent.obj
 .catalogues.InvalidInputException: Campo: metodoDiPagamento è vuoto
> <5 internal lines>
    at blackbox.RegistraPrenotazione.TestCasesRegistraPrenotazione$TestPassRegistraPrenotazione
>    .testCase4(TestCasesRegistraPrenotazione.java:226) <1 internal line>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1604)
><2 folded frames>
Caused by: it.unisa.Server.persistent.obj.catalogues.InvalidInputException Create breakpoint : Campo:
 metodoDiPagamento è vuoto
    at it.unisa.Server.persistent.obj.catalogues.CatalogueUtils.checkNull(CatalogueUtils.java:29)
    at it.unisa.Server.gestionePrenotazioni.FrontDesk.addPrenotazione(FrontDesk.java:117)
>     <6 internal lines>
><13 folded frames>
```

Test Case	Test Result
TC1	**FAIL**
TC2	**FAIL**
TC3	**FAIL**
TC4	**FAIL**
TC5	**PASS**
TC6	**PASS**
TC7	**PASS**
TC8	**PASS**
TC9	**PASS**
TC10	**PASS**
TC11	**PASS**
TC12	**PASS**
TC13	**PASS**
TC14	**PASS**
TC15	**PASS**
TC16	**PASS**
TC17	**PASS**
TC18	**PASS**
TC19	**PASS**
TC20	**PASS**
TC21	**PASS**
TC22	**PASS**
TC23	**PASS**
TC24	**PASS**

TEST RESULT: WHITE BOX

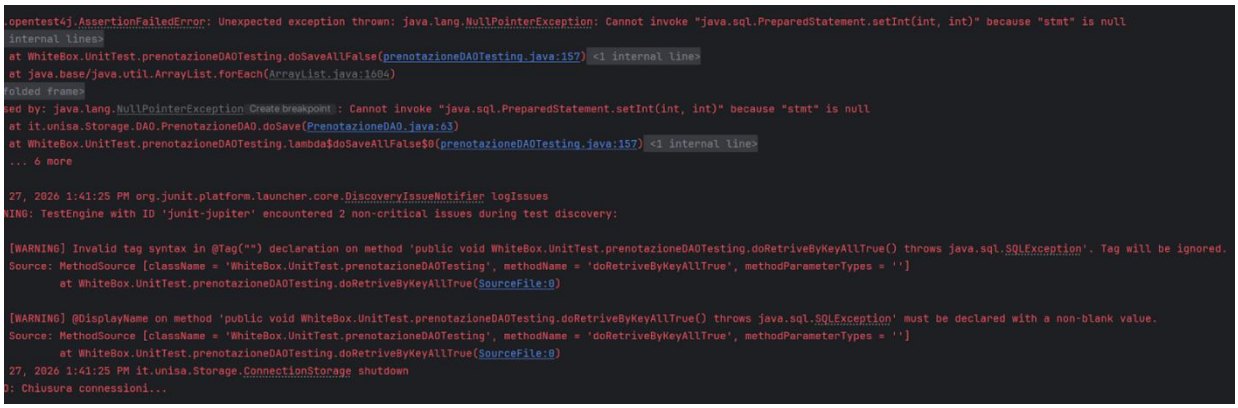
TEST DI UNITÀ: WHITEBOX

Titolo: Test di salvataggio prenotazione con NullPointerException su trattamento

Descrizione dell'errore: Il test ha generato una NullPointerException durante l'esecuzione del metodo doSaveAllFalse(). L'errore si è verificato perché nel codice è stato invocato il metodo p.getTrattamento().getNome() senza verificare preventivamente se l'oggetto Trattamento fosse diverso da null. Inoltre, non è stato effettuato alcun controllo sulla validità del nome del trattamento. Quando il PreparedStatement ha tentato di eseguire setString() con un valore null derivante da un oggetto trattamento non inizializzato, il sistema ha lanciato l'eccezione Cannot invoke "PreparedStatement.setInt(int, int)" because "stmt" is null.

Foto:

- **Errore riscontrato (Test Failure):**



- **Codice causa dell'errore (Bugged Code):**

```
preparedStatement.setString( parameterIndex: 10, p.getTrattamento().getNome());
```

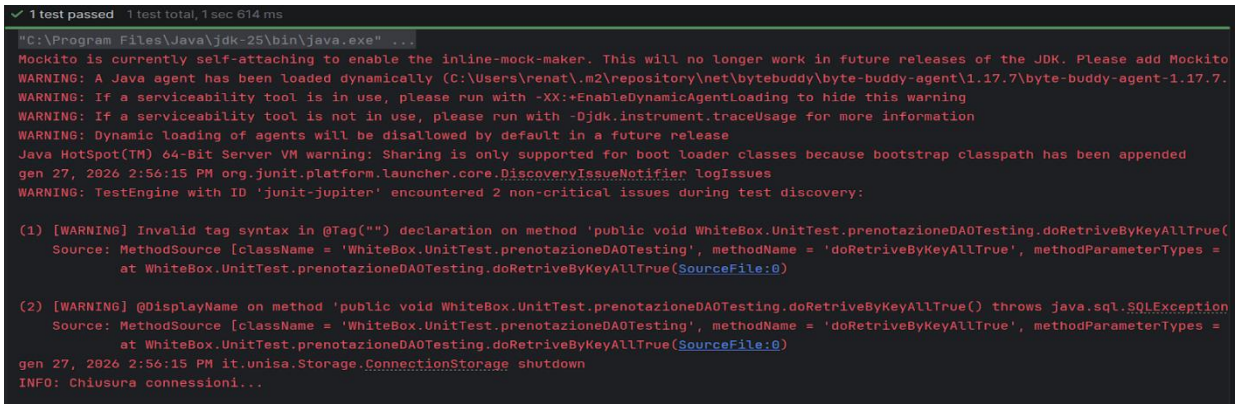
- **Fix applicato (Corrected Code):**

```
if(p != null && p.getTrattamento() != null){
    Connection connection = ConnectionStorage.getConnection();
    PreparedStatement preparedStatement = connection.prepareStatement( sql: "INSERT INTO prenota
            "VALUES (?, ?, ?, ?, ?, ?, ? , ?, ? , ? , ? , ?)");

    preparedStatement.setDate( parameterIndex: 1,Date.valueOf(p.getDataCreazionePrenotazione()));
    preparedStatement.setDate( parameterIndex: 2,Date.valueOf(p.getDataInizio()));
    preparedStatement.setDate( parameterIndex: 3,Date.valueOf(p.getDataFine()));
    preparedStatement.setString( parameterIndex: 10, p.getTrattamento().getNome());
    preparedStatement.setString( parameterIndex: 4,p.getNoteAggiuntive());
    preparedStatement.setString( parameterIndex: 5 , p.getIntestatario());
    preparedStatement.setDate( parameterIndex: 6, Date.valueOf(p.getDataScadenza()));
    preparedStatement.setString( parameterIndex: 7,p.getNumeroDocumento());
    preparedStatement.setDate( parameterIndex: 8,Date.valueOf(p.getDataRilascio()));
    preparedStatement.setString( parameterIndex: 9,p.getTipoDocumento());
    preparedStatement.setBoolean( parameterIndex: 11,p.getStatoPrenotazione());
    preparedStatement.setBoolean( parameterIndex: 12,p.isCheckIn());
    preparedStatement.executeUpdate();

    // Salva il trattamento associato
    if(p.getTrattamento().getNome() != null){
        String query = "UPDATE Trattamento SET IDPrenotazione = ? WHERE Nome = ?";

        try (PreparedStatement stmt = connection.prepareStatement(query)) {
            stmt.setInt( parameterIndex: 1, p.getIDPrenotazione());
            stmt.setString( parameterIndex: 2, p.getTrattamento().getNome());
            stmt.executeUpdate();
        }
    }
```

- **Metodo di test**

```
@Test  Renato
@Tag("False")
@DisplayName("doSave() quando è tutto False")
public void doSaveAllFalse() throws SQLException{
    prenotazione.setListaServizi(new ArrayList<>());
    prenotazione.setListaCamere(new ArrayList<>());
    prenotazione.setListaClienti(new ArrayList<>());
    assertDoesNotThrow(()->prenotazioneDAO.doSave(prenotazione));
}
```

- **Test passato (Success):**



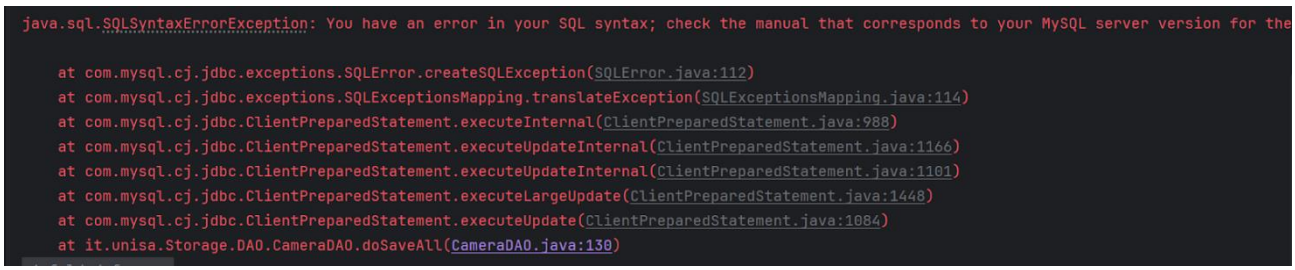
TEST DI UNITÀ: WHITEBOX

Titolo: Test di salvataggio multiplo camere con SQLSyntaxErrorException

Descrizione dell'errore: Il test ha generato una SQLSyntaxErrorException. L'errore si è verificato perché nel codice non viene controllata la dimensione (size) della lista listCamera prima di costruire dinamicamente la query SQL di INSERT. Questo ha causato la creazione di una query malformata con una sintassi SQL non corretta. Il ciclo for che costruisce la stringa VALUES iniziava da i = 1 invece che da i = 0, e il controllo per aggiungere la virgola o il punto e virgola era basato su un confronto errato (i == numCamere invece di verificare l'ultimo elemento effettivo). Inoltre, mancava completamente un controllo preventivo per verificare se la lista fosse vuota prima di procedere con l'operazione di inserimento.

Foto:

- **Errore riscontrato (Test Failure):**



- **Codice causa dell'errore (Bugged Code):**

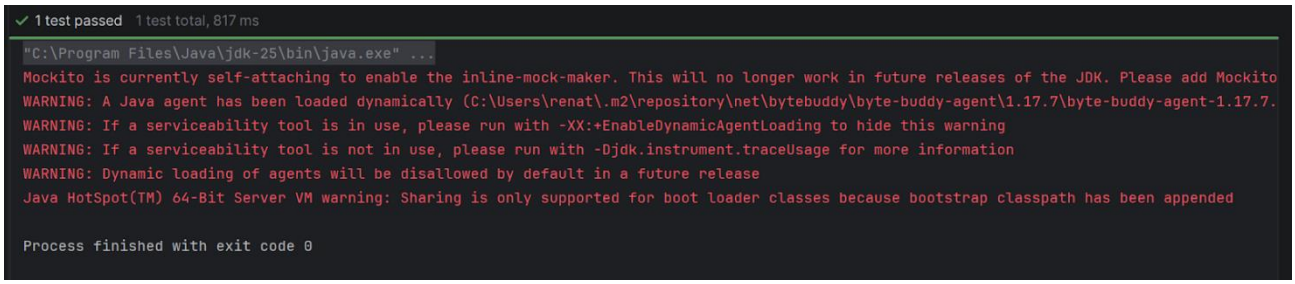
```
@Override  1 usage  ± RccGnn
public synchronized void doSaveAll(List<Camera> listCamera) throws SQLException {
    StringBuilder insertSQL = new StringBuilder();
    String values = " (?, ?, ?, ?, ?) ";
    insertSQL.append("INSERT INTO " + CameraDAO.TABLE_NAME + " VALUES ");
    int numCamere = listCamera.size(); // numCamere * 5 = numCampi ?

    // Crea la query con i
    for(int i = 1; i <= numCamere; i++){
        insertSQL.append(values);
        if(i == numCamere){
            insertSQL.append(";");
        } else {
            insertSQL.append(",");
        }
    }

    // Riempi la query
    connection = ConnectionStorage.getConnection();
    PreparedStatement preparedStatement = connection.prepareStatement(insertSQL.toString());
    for (int i = 0; i < numCamere; i++) {
        Camera c = listCamera.get(i);
        preparedStatement.setInt( parameterIndex: 1 + 5*i, c.getNumeroCamera());
        preparedStatement.setInt( parameterIndex: 2 + 5*i, c.getCapacità());
        preparedStatement.setString( parameterIndex: 3 + 5*i, c.getNoteCamera());
        preparedStatement.setObject( parameterIndex: 4 + 5*i, c.getStatoCamera().name());
        preparedStatement.setDouble( parameterIndex: 5 + 5*i, c.getPrezzoCamera());
    }

    try{
        preparedStatement.executeUpdate();
    } finally {
        if(connection != null){
            ConnectionStorage.releaseConnection(connection);
        }
    }
}
```

- **Fix applicato (Corrected Code):**
- **Metodo di test**

```java
public synchronized void doSaveAll(List<Camera> listCamera) throws SQLException {
    if(!listCamera.isEmpty()){
        StringBuilder insertSQL = new StringBuilder();
        String values = " (?, ?, ?, ?, ?) ";
        insertSQL.append("INSERT INTO " + CameraDAO.TABLE_NAME + " VALUES ");
        int numCamere = listCamera.size(); // numCamere * 5 = numCampi ?

        // Crea la query con i
        for(int i = 1; i <= numCamere; i++){
            insertSQL.append(values);
            if(i == numCamere){
                insertSQL.append(";");
            } else {
                insertSQL.append(",");
            }
        }

        // Riempi la query
        connection = ConnectionStorage.getConnection();
        PreparedStatement preparedStatement = connection.prepareStatement(insertSQL.toString(
        for(int i = 0; i < numCamere; i++){
            Camera c = listCamera.get(i);
            preparedStatement.setInt( parameterIndex: 1 + 5*i, c.getNumeroCamera());
            preparedStatement.setInt( parameterIndex: 2 + 5*i, c.getCapacità());
            preparedStatement.setString( parameterIndex: 3 + 5*i, c.getNoteCamera());
            preparedStatement.setObject( parameterIndex: 4 + 5*i, c.getStatoCamera().name());
            preparedStatement.setDouble( parameterIndex: 5 + 5*i, c.getPrezzoCamera());
        }

        try{
            preparedStatement.executeUpdate();
        } finally {
            if(connection != null){
                ConnectionStorage.releaseConnection(connection);
            }
        }
    }else{
        throw new NullPointerException("la lista è null oppure la dimensione è uguale a 0");
```

```java
@Test  Renato
@Tag("True")
@DisplayName("doSaveAll() quando è tutto vero")
public void doSaveAllTrue(){
    ArrayList<Camera> cameras = new ArrayList<>();
    cameras.add(camera);
    cameras.add(new Camera( numeroCamera: 105,Stato.Libera, capacità: 3, prezzoCamera: 155.0, noteCamera: "Sp"));
}
```

- **Test passato (Success):**



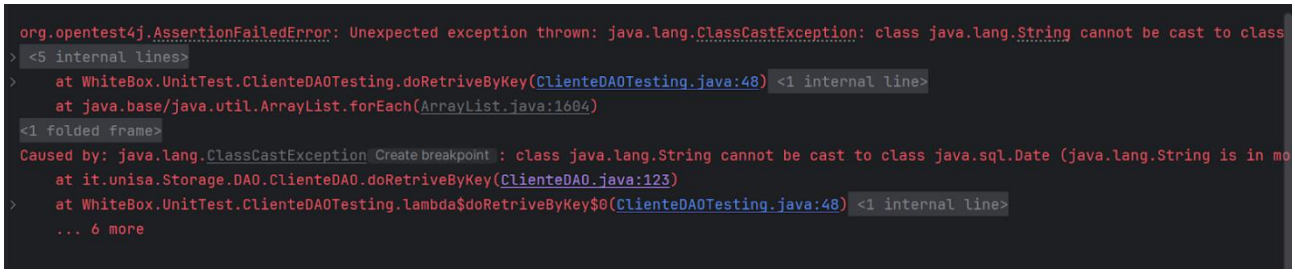
TEST DI UNITÀ: WHITEBOX

Titolo: Test di recupero cliente con ClassCastException su conversione Date

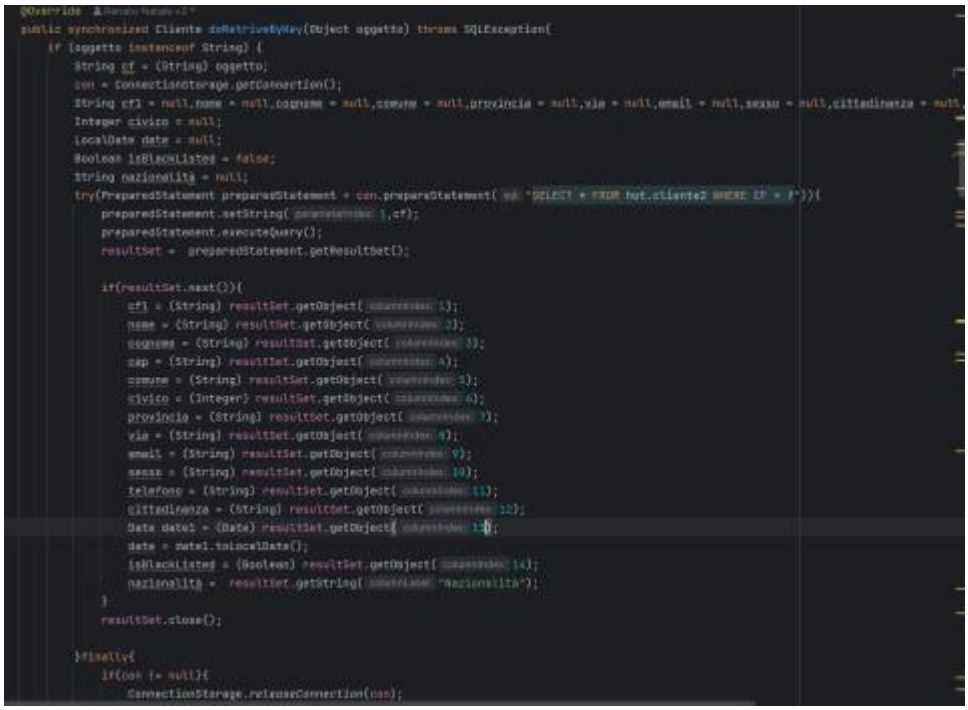
Descrizione dell'errore: Il test ha generato una ClassCastException con il messaggio "class java.lang.String cannot be cast to class java.sql.Date". L'errore si è verificato nel metodo doRetrieveByKey. Il problema è stato causato da un cast errato quando si tentava di convertire il valore recuperato dal ResultSet in un oggetto Date. A causa di eventuali modifiche nella struttura del database, la colonna che si presumeva contenesse un tipo Date restituiva invece un tipo String, rendendo impossibile il cast diretto a Date. Il codice non gestiva questa possibilità e tentava forzatamente di eseguire il cast senza validazione del tipo effettivo del dato.

Foto:

Errore riscontrato (Test Failure):



- **Codice causa dell'errore (Bugged Code):**

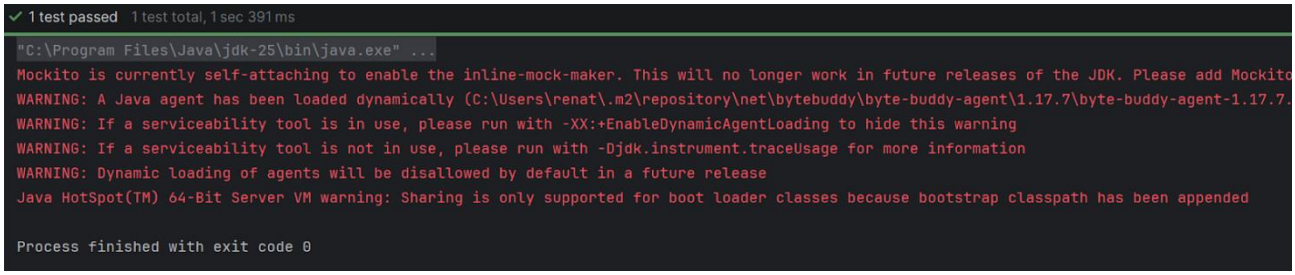


- **Fix applicato (Corrected Code):**
- **Metodo di test**

```
@Override  Renato Natale +2 *
public synchronized Cliente doRetriveByKey(Object oggetto) throws SQLException{
    if (oggetto instanceof String) {
        String cf = (String) oggetto;
        con = ConnectionStorage.getConnection();
        String cf1 = null,nome = null,cognome = null,comune = null,provincia = null,via = null,email = null,sesso = null,cittadinanza = null,telefono = null , cap = null;
        Integer civico = null;
        LocalDate date = null;
        Boolean isBlackListed = false;
        String nazionalità = null;
        try(PreparedStatement preparedStatement = con.prepareStatement( sql: "SELECT * FROM hot.cliente2 WHERE CF = ?")){
            preparedStatement.setString( parameterIndex: 1,cf);
            preparedStatement.executeQuery();
            resultSet =  preparedStatement.getResultSet();

            if(resultSet.next()){
                cf1 = (String) resultSet.getObject( columnLabel: "CF");
                nome = (String) resultSet.getObject( columnLabel: "nome");
                cognome = (String) resultSet.getObject( columnLabel: "cognome");
                cap = (String) resultSet.getObject( columnLabel: "Cap");
                comune = (String) resultSet.getObject( columnLabel: "comune");
                civico = (Integer) resultSet.getObject( columnLabel: "civico");
                provincia = (String) resultSet.getObject( columnLabel: "provincia");
                via = (String) resultSet.getObject( columnLabel: "via");
                email = (String) resultSet.getObject( columnLabel: "Email");
                sesso = (String) resultSet.getObject( columnLabel: "Sesso");
                telefono = (String) resultSet.getObject( columnLabel: "telefono");
                cittadinanza = (String) resultSet.getObject( columnLabel: "Cittadinanza");
                Date date1 = (Date) resultSet.getObject( columnLabel: "DataDiNascità");
                date = date1.toLocalDate();
                isBlackListed = (Boolean) resultSet.getObject( columnLabel: "IsBackListed");
                nazionalità =  resultSet.getString( columnLabel: "Nazionalità");
            }
            resultSet.close();

        }finally{
```

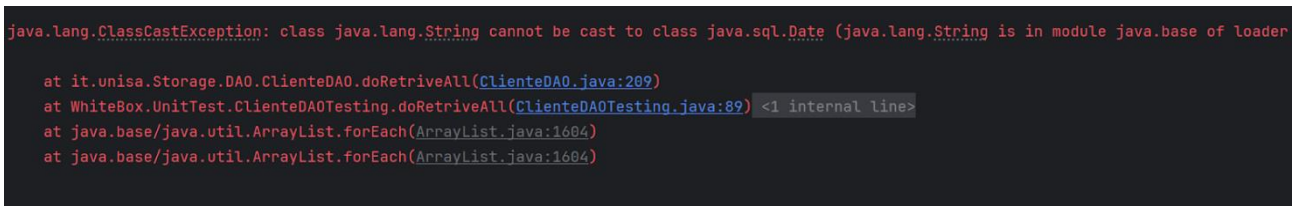
```java
@Test  Renato
@Tag("True")
@DisplayName("doRetriveByKey() quando va tutto bene")
public void doRetriveByKeyAllTrue(){
    assertDoesNotThrow(()->clienteDAO.doRetriveByKey( oggetto: "RSSMRA20T11H703F"));
}
```

- **Test passato (Success):**



TEST DI UNITÀ: WHITEBOX
Titolo: Test di recupero multiplo clienti con ClassCastException su conversione Date
Descrizione dell'errore: Il test ha generato una ClassCastException con il messaggio "class java.lang.String cannot be cast to class java.sql.Date". L'errore si è verificato nel metodo doRetrieveAll(). Il problema è stato causato da un tentativo di cast errato quando si cercava di recuperare la data di nascita dal ResultSet utilizzando gli indici numerici delle colonne. A causa di eventuali modifiche nella struttura del database, la colonna che si presumeva contenesse un tipo Date restituiva invece un tipo String, rendendo impossibile il cast diretto a Date.
Foto:

- **Errore riscontrato (Test Failure):**



- **Codice causa dell'errore (Bugged Code):**

```java
@Override  Renato Natale +2 *
public synchronized Collection<Cliente> doRetriveAll(String order) throws SQLException {
    con = ConnectionStorage.getConnection();
    ArrayList<Cliente> clientes = new ArrayList<>();

    String sql =  "SELECT * FROM hot.cliente2 ORDER BY ? ";
    if(order != null){
        if(order.equalsIgnoreCase( anotherString: "decrescente")){
            sql += "DESC";
        }else{
            sql+= "ASC";
        }
    }

    try(PreparedStatement preparedStatement = con.prepareStatement(sql)){
        preparedStatement.setString( parameterIndex: 1, x: "CF");
        ResultSet resultSet = preparedStatement.executeQuery();
        while(resultSet.next()){
            String cf1 = (String) resultSet.getObject( columnIndex: 1);
            String nome = (String) resultSet.getObject( columnIndex: 2);
            String cognome = (String) resultSet.getObject( columnIndex: 3);
            String  cap = (String) resultSet.getObject( columnIndex: 4);
            String comune = (String) resultSet.getObject( columnIndex: 5);
            Integer civico = (Integer) resultSet.getObject( columnIndex: 6);
            String provincia = (String) resultSet.getObject( columnIndex: 7);
            String via = (String) resultSet.getObject( columnIndex: 8);
            String email = (String) resultSet.getObject( columnIndex: 9);
            String  sesso = (String) resultSet.getObject( columnIndex: 10);
            String  telefono = (String) resultSet.getObject( columnIndex: 11);
            String  cittadinazione = (String) resultSet.getObject( columnIndex: 12);
            Date date1 = (Date) resultSet.getObject( columnIndex: 13);
            LocalDate date = date1.toLocalDate();
            Boolean  isBackListed = (Boolean) resultSet.getObject( columnIndex: 14);
            String nazionalità = resultSet.getString( columnLabel: "Nazionalità");
            cliente = new Cliente(nome,cognome,cittadinazione,provincia,comune,via,civico,Integer.parseInt(cap),telefono,sesso,date,cf1,email,nazionalità);
            cliente.setBlacklisted(isBackListed);
            clientes.add(cliente);
        }
```

- **Fix applicato (Corrected Code):**
- **Metodo di test**

```java
@Override  Renato Natale +2 *
public synchronized Collection<Cliente> doRetriveAll(String order) throws SQLException {
    con = ConnectionStorage.getConnection();
    ArrayList<Cliente> clientes = new ArrayList<>();

    String sql =  "SELECT * FROM hot.cliente2 ORDER BY ? ";
    if(order != null){
        if(order.equalsIgnoreCase( anotherString: "decrescente")){
            sql += "DESC";
        }else{
            sql+= "ASC";
        }
    }

    try(PreparedStatement preparedStatement = con.prepareStatement(sql)){
        preparedStatement.setString( parameterIndex: 1, x: "CF");
        ResultSet resultSet = preparedStatement.executeQuery();
        while(resultSet.next()){
            String cf1 = (String) resultSet.getObject( columnLabel: "CF");
            String nome = (String) resultSet.getObject( columnLabel: "Nome");
            String cognome = (String) resultSet.getObject( columnLabel: "Cognome");
            String  cap = (String) resultSet.getObject( columnLabel: "Cap");
            String comune = (String) resultSet.getObject( columnLabel: "comune");
            Integer civico = (Integer) resultSet.getObject( columnLabel: "civico");
            String provincia = (String) resultSet.getObject( columnLabel: "provincia");
            String via = (String) resultSet.getObject( columnLabel: "via");
            String email = (String) resultSet.getObject( columnLabel: "email");
            String  sesso = (String) resultSet.getObject( columnLabel: "Sesso");
            String  telefono = (String) resultSet.getObject( columnLabel: "telefono");
            String  cittadinazione = (String) resultSet.getObject( columnLabel: "Cittadinanza");
            Date date1 = (Date) resultSet.getObject( columnLabel: "DataDiNascita");
            LocalDate date = date1.toLocalDate();
            Boolean  isBackListed = (Boolean) resultSet.getObject( columnLabel: "IsBackListed");
            String nazionalità = resultSet.getString( columnLabel: "Nazionalità");
            cliente = new Cliente(nome,cognome,cittadinazione,provincia,comune,via,civico,Integer.parseInt(cap),telefono,sesso,date,cf1,email,nazionalità);
            cliente.setBlacklisted(isBackListed);
            clientes.add(cliente);
        }
```

```java
@Test  Renato *
@Tag("True")
@DisplayName("doRetriveAll() quando True e quindi è decrescente")
public void doRetriveAllTrue() throws SQLException {
    ArrayList<Cliente> clientes;
    ArrayList<Cliente> clientes1 = new ArrayList<>();
    clientes = (ArrayList<Cliente>) clienteDAO.doRetriveAll( order: "decrescente");

   // clientes1.add(new Cliente("Mario","Rossi","Italiana","Napoli","Napoli","Via Roma",15,80100,"3331234567","Maschio",LocalDate.of(
   // clientes1.add(new Cliente("Laura","Verdi","Italiana","Roma","Roma","Via Milano",23,100,"3339876543","Femmina",LocalDate.of(1990

    assertEquals(clientes1,clientes);
}
```

- **Test passato (Success):**

```
✓ 1 test passed  1 test total, 994 ms
"C:\Program Files\Java\jdk-25\bin\java.exe" ...
Mockito is currently self-attaching to enable the inline-mock-maker. This will no longer work in future releases of the JDK. Please add Mockito
WARNING: A Java agent has been loaded dynamically (C:\Users\renat\.m2\repository\net\bytebuddy\byte-buddy-agent\1.17.7\byte-buddy-agent-1.17.7.
WARNING: If a serviceability tool is in use, please run with -XX:+EnableDynamicAgentLoading to hide this warning
WARNING: If a serviceability tool is not in use, please run with -Djdk.instrument.traceUsage for more information
WARNING: Dynamic loading of agents will be disallowed by default in a future release
Java HotSpot(TM) 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended

Process finished with exit code 0
```

TEST DI UNITÀ: WHITEBOX

Titolo: Test di recupero clienti per attributo con AssertionFailedError su valore null

Descrizione dell'errore: Il test ha generato un AssertionFailedError indicando una differenza tra il risultato atteso e quello effettivo . L'errore mostra che entrambe le liste contengono lo stesso cliente ma il test fallisce comunque. Il problema è stato causato dal fatto che nel metodo doRetrieveByAttribute() manca l'inizializzazione e il recupero del campo "Nazionalità" dal ResultSet. Questo campo viene lasciato a null nell'oggetto Cliente creato, mentre il cliente atteso nel test ha questo campo valorizzato. Anche se tutti gli altri campi corrispondono perfettamente, la presenza di un singolo campo null causa il fallimento dell'assertion di uguaglianza tra gli oggetti, poiché il metodo equals() della classe Cliente considera tutti i campi nella comparazione.

Foto:

- **Errore riscontrato (Test Failure):**

```
org.opentest4j.AssertionFailedError:
Expected :[Cliente{nome='Mario', cf='RSSMRA85M01H501Z', email='mario.rossi@email.it', cognome='Rossi', cittadinanza='Italiana', provincia='Napo

Actual   :[Cliente{nome='Mario', cf='RSSMRA85M01H501Z', email='mario.rossi@email.it', cognome='Rossi', cittadinanza='Italiana', provincia='Napo

<Click to see difference>

> <6 internal lines>
>     at WhiteBox.UnitTest.ClienteDAOTesting.doRetriveByAttributeAllTrue(ClienteDAOTesting.java:153) <1 internal line>
><2 folded frames>
```

- **Codice causa dell'errore (Bugged Code):**

```java
public synchronized Collection<Cliente> doRetriveByAttribute(String attribute, Object value) throws SQLExceptic
    PreparedStatement preparedStatement = null;
    ArrayList<Cliente> lista = new ArrayList<>();
    String selectSQL;

    if(attribute != null && !attribute.isEmpty() && value != null){
        con= ConnectionStorage.getConnection();
        selectSQL = "SELECT * FROM hot.cliente2 WHERE " + attribute + " = ?";
        try{
            preparedStatement = con.prepareStatement(selectSQL);
            preparedStatement.setObject( parameterIndex: 1, value);
            ResultSet resultSet = preparedStatement.executeQuery();

            while (resultSet.next()) {
                cliente = new Cliente();
                cliente.setCf(resultSet.getString( columnLabel: "CF"));
                cliente.setNome(resultSet.getString( columnLabel: "nome"));
                cliente.setCognome(resultSet.getString( columnLabel: "cognome"));
                cliente.setNumeroCivico(resultSet.getInt( columnLabel: "civico"));
                cliente.setCAP(resultSet.getInt( columnLabel: "Cap"));
                cliente.setComune(resultSet.getString( columnLabel: "Comune"));
                cliente.setCittadinanza(resultSet.getString( columnLabel: "Cittadinanza"));
                cliente.setProvincia(resultSet.getString( columnLabel: "provincia"));
                cliente.setVia(resultSet.getString( columnLabel: "Via"));
                cliente.setEmail(resultSet.getString( columnLabel: "Email"));
                cliente.setSesso(resultSet.getString( columnLabel: "Sesso"));
                cliente.setNumeroTelefono(resultSet.getString( columnLabel: "telefono"));
                cliente.setBlacklisted(resultSet.getBoolean( columnLabel: "IsBackListed"));
                cliente.setDataNascita(resultSet.getDate( columnLabel: "DataDiNascita").toLocalDate());
                lista.add(cliente);
            }

        }finally{
            if(con!= null){
                if (preparedStatement != null) {
                    preparedStatement.close();
                }
```

- **Fix applicato (Corrected Code):**
- **Metodo di test**

```java
@Override  RccGnn +2 *
public synchronized Collection<Cliente> doRetriveByAttribute(String attribute, Object value) throws SQLException {
    PreparedStatement preparedStatement = null;
    ArrayList<Cliente> lista = new ArrayList<>();
    String selectSQL;

    if(attribute != null && !attribute.isEmpty() && value != null){
        con= ConnectionStorage.getConnection();
        selectSQL = "SELECT * FROM hot.cliente2 WHERE " + attribute + " = ?";
        try{
            preparedStatement = con.prepareStatement(selectSQL);
            preparedStatement.setObject( parameterIndex: 1, value);
            ResultSet resultSet = preparedStatement.executeQuery();

            while (resultSet.next()) {
                cliente = new Cliente();
                cliente.setCf(resultSet.getString( columnLabel: "CF"));
                cliente.setNome(resultSet.getString( columnLabel: "nome"));
                cliente.setCognome(resultSet.getString( columnLabel: "cognome"));
                cliente.setNumeroCivico(resultSet.getInt( columnLabel: "civico"));
                cliente.setCAP(resultSet.getInt( columnLabel: "Cap"));
                cliente.setComune(resultSet.getString( columnLabel: "Comune"));
                cliente.setCittadinanza(resultSet.getString( columnLabel: "Cittadinanza"));
                cliente.setProvincia(resultSet.getString( columnLabel: "provincia"));
                cliente.setVia(resultSet.getString( columnLabel: "Via"));
                cliente.setEmail(resultSet.getString( columnLabel: "Email"));
                cliente.setSesso(resultSet.getString( columnLabel: "Sesso"));
                cliente.setNumeroTelefono(resultSet.getString( columnLabel: "telefono"));
                cliente.setBlacklisted(resultSet.getBoolean( columnLabel: "IsBackListed"));
                cliente.setDataNascita(resultSet.getDate( columnLabel: "DataDiNascita").toLocalDate());
                cliente.setNazionalita(resultSet.getString( columnLabel: "Nazionalità"));
                lista.add(cliente);
            }
```

```java
@Test  Renato *
@Tag("True")
@DisplayName("doRetriveByAttribute() quando va tutto bene")
public void doRetriveByAttributeAllTrue() throws SQLException {
    ArrayList<Cliente> clientes = new ArrayList<>();
   // clientes.add(new Cliente("Mario","Rossi","Italiana","Napoli","Napoli","Via Roma",15,80100,"3331234567","Maschio",LocalDate.of(
   // clientes.add(new Cliente("Laura","Verdi","Italiana","Roma","Roma","Via Milano",23,100,"3339876543","Femmina",LocalDate.of(19
    Object s = "Italiana" ;
    ArrayList<Cliente> clientes1 = (ArrayList<Cliente>) clienteDAO.doRetriveByAttribute( attribute: "Cittadinanza",s);
    assertEquals(clientes,clientes1);
}
```

- **Test passato (Success):**

```
✓ 1 test passed  1 test total, 1 sec 398 ms

"C:\Program Files\Java\jdk-25\bin\java.exe" ...
Mockito is currently self-attaching to enable the inline-mock-maker. This will no longer work in future releases of the JDK. Please add Mockito
WARNING: A Java agent has been loaded dynamically (C:\Users\renat\.m2\repository\net\bytebuddy\byte-buddy-agent\1.17.7\byte-buddy-agent-1.17.7.
WARNING: If a serviceability tool is in use, please run with -XX:+EnableDynamicAgentLoading to hide this warning
WARNING: If a serviceability tool is not in use, please run with -Djdk.instrument.traceUsage for more information
WARNING: Dynamic loading of agents will be disallowed by default in a future release
Java HotSpot(TM) 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended

Process finished with exit code 0
```

TEST DI UNITÀ: WHITEBOX

Titolo: Test di filtraggio clienti con AssertionFailedError su campi mancanti (nazionalità e isBlackListed)

Descrizione dell'errore: Il test ha generato un AssertionFailedError indicando una differenza tra il risultato atteso e quello effettivo. L'errore mostra che le due liste contengono apparentemente lo stesso cliente, ma differiscono in realtà per due campi specifici. Nel metodo doFilter() della classe, mancava il recupero di due campi fondamentali dal ResultSet: la "Nazionalità" che rimaneva null nell'oggetto creato mentre nel cliente atteso era "Italiana", e il campo "isBlackListed" che non veniva impostato, risultando null invece di false. Anche se tutti gli altri campi corrispondevano perfettamente, la mancanza di questi due valori causava il fallimento dell'assertion di uguaglianza tra gli oggetti Cliente, poiché il metodo equals() considera tutti i campi nella comparazione.

Foto:

- **Errore riscontrato (Test Failure):**

- **Codice causa dell'errore (Bugged Code):**

```
            }
            resultSet = preparedStatement.executeQuery();

            while (resultSet.next()){
                cliente = new Cliente();
                cliente.setCf(resultSet.getString( columnLabel: "CF"));
                cliente.setNome(resultSet.getString( columnLabel: "nome"));
                cliente.setCognome(resultSet.getString( columnLabel: "cognome"));
                cliente.setCAP(resultSet.getInt( columnLabel: "Cap"));
                cliente.setComune(resultSet.getString( columnLabel: "comune"));
                cliente.setNumeroCivico(resultSet.getInt( columnLabel: "civico"));
                cliente.setProvincia(resultSet.getString( columnLabel: "provincia"));
                cliente.setVia(resultSet.getString( columnLabel: "via"));
                cliente.setEmail(resultSet.getString( columnLabel: "Email"));
                cliente.setSesso(resultSet.getString( columnLabel: "Sesso"));
                cliente.setNumeroTelefono(resultSet.getString( columnLabel: "telefono"));
                cliente.setCittadinanza(resultSet.getString( columnLabel: "Cittadinanza"));

                lista.add(cliente);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            if (preparedStatement != null)
                preparedStatement.close();
            ConnectionStorage.releaseConnection(con);
        }
```

- **Fix applicato (Corrected Code):**

```
            counter++;
        }
        if (params[2]) {
            preparedStatement.setString(counter, nazionalita);
            counter++;
        }
        if (params[3]) {
            preparedStatement.setDate(counter, Date.valueOf(dataNascita));
            counter++;
        }
        if (params[4]) {
            preparedStatement.setBoolean(counter, blackListed);
        }
        resultSet = preparedStatement.executeQuery();

        while (resultSet.next()){
            cliente = new Cliente();
            cliente.setCf(resultSet.getString( columnLabel: "CF"));
            cliente.setNome(resultSet.getString( columnLabel: "nome"));
            cliente.setCognome(resultSet.getString( columnLabel: "cognome"));
            cliente.setCAP(resultSet.getInt( columnLabel: "Cap"));
            cliente.setComune(resultSet.getString( columnLabel: "comune"));
            cliente.setNumeroCivico(resultSet.getInt( columnLabel: "civico"));
            cliente.setProvincia(resultSet.getString( columnLabel: "provincia"));
            cliente.setVia(resultSet.getString( columnLabel: "via"));
            cliente.setEmail(resultSet.getString( columnLabel: "Email"));
            cliente.setSesso(resultSet.getString( columnLabel: "Sesso"));
            cliente.setNumeroTelefono(resultSet.getString( columnLabel: "Telefono"));
            cliente.setCittadinanza(resultSet.getString( columnLabel: "Cittadinanza"));
            cliente.setBlacklisted(resultSet.getBoolean( columnLabel: "IsBackListed"));
            cliente.setNazionalita(resultSet.getString( columnLabel: "Nazionalita"));
            cliente.setDataNascita(resultSet.getDate( columnLabel: "DataDiNascita").toLocalDate());

            lista.add(cliente);
```

- **Metodo di test**

```
@Test  Renato *
@Tag("True")
@DisplayName("doFilter() quando va tutto bene")
public void doFilterAllTrue(){
    ArrayList<Cliente> clientes1 = new ArrayList<>();
    ArrayList<Cliente> clientes = (ArrayList<Cliente>) clienteDAO.doFilter( nome: "", cognome: "", nazionalita: "",LocalDate.of( year: 1985
   // clientes1.add(new Cliente("Mario","Rossi","Italiana","Napoli","Napoli","Via Roma",15,80100,"3331234567","Maschio",LocalDate.of
    assertEquals(clientes,clientes1);
}
```

- **Test passato (Success):**

```
✓ 1 test passed  1 test total, 1 sec 534 ms
"C:\Program Files\Java\jdk-25\bin\java.exe" ...
Mockito is currently self-attaching to enable the inline-mock-maker. This will no longer work in future releases of the JDK. Please add Mockito
WARNING: A Java agent has been loaded dynamically (C:\Users\renat\.m2\repository\net\bytebuddy\byte-buddy-agent\1.17.7\byte-buddy-agent-1.17.7.
WARNING: If a serviceability tool is in use, please run with -XX:+EnableDynamicAgentLoading to hide this warning
WARNING: If a serviceability tool is not in use, please run with -Djdk.instrument.traceUsage for more information
WARNING: Dynamic loading of agents will be disallowed by default in a future release
Java HotSpot(TM) 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended

Process finished with exit code 0
```

TEST DI UNITÀ: WHITEBOX

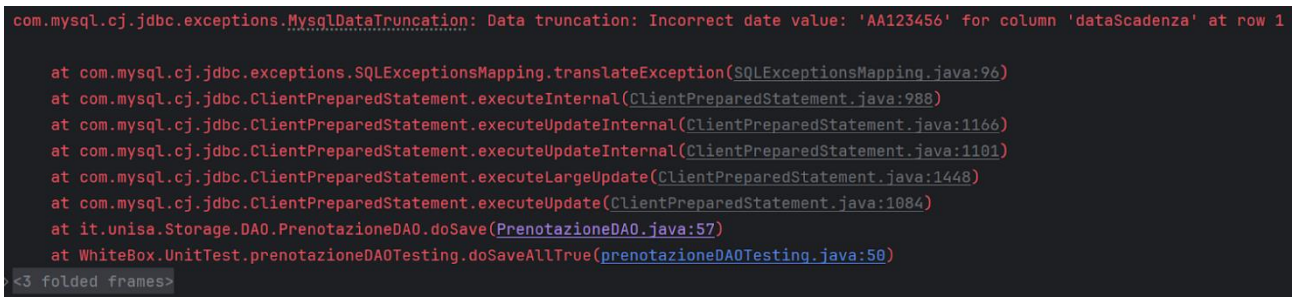
Titolo: Test di salvataggio prenotazione con MysqlDataTruncation su formato data errato

Descrizione dell'errore: Il test ha generato una MysqlDataTruncation con il messaggio "Data truncation: Incorrect date value: 'AA123456' for column 'dataScadenza' at row 1". L'errore si è verificato nel metodo doSave() durante l'esecuzione del executeUpdate().

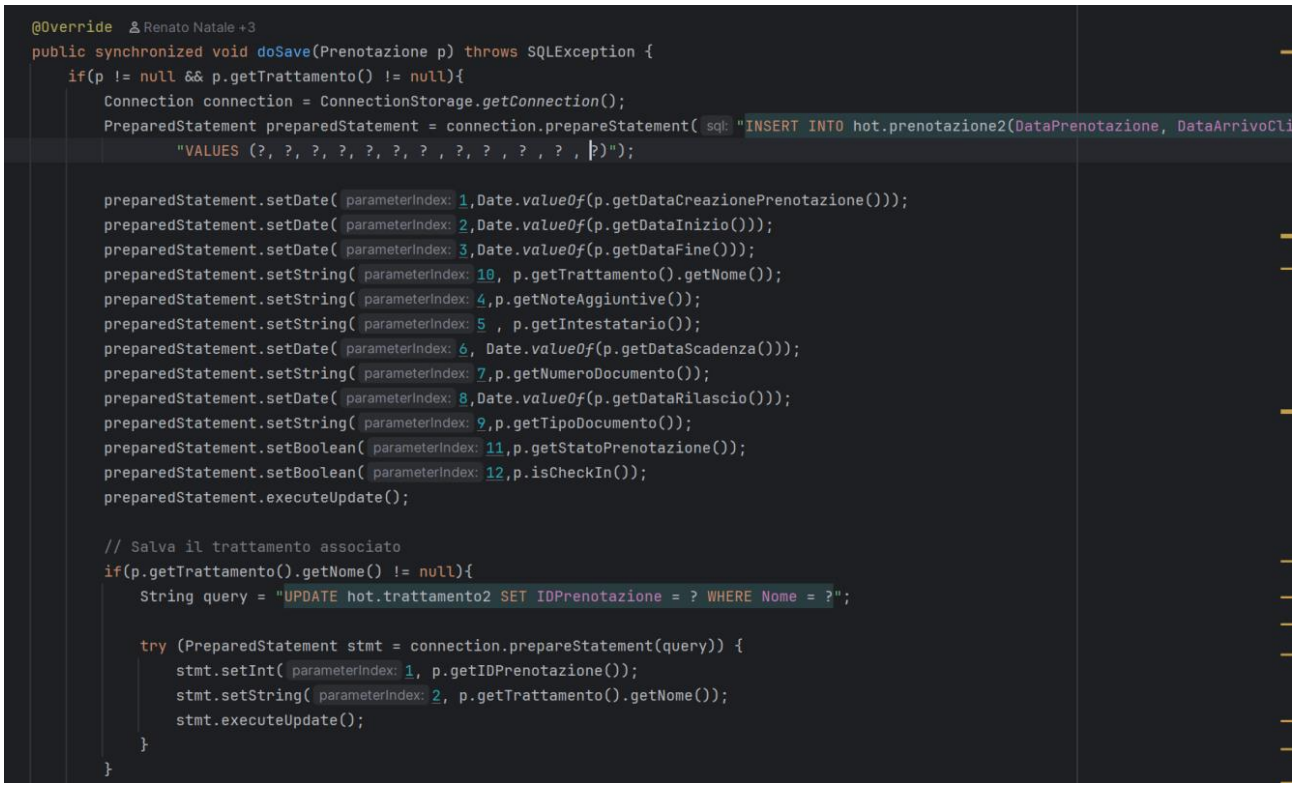
Il problema è stato causato da un errore nell'ordine dei parametri impostati nel PreparedStatement. Il codice stava erroneamente assegnando il valore di getDataScadenza() usando valueOf(), ma in realtà stava passando un valore non-data (probabilmente una stringa come "AA123456") alla colonna del database che si aspettava un tipo DATE. Questo mismatch tra il tipo di dato atteso dal database e il valore effettivamente fornito ha causato il troncamento dei dati e l'errore di conversione. L'ordine scorretto dei parametri nel PreparedStatement faceva sì che i valori venissero inseriti nelle colonne sbagliate, con conseguente tentativo di inserire dati incompatibili.

Foto:

- **Errore riscontrato (Test Failure):**



- **Codice causa dell'errore (Bugged Code):**



- **Fix applicato (Corrected Code):**
- **Metodo di test**

```
public synchronized void doSave(Prenotazione p) throws SQLException {
    if(p != null && p.getTrattamento() != null){
        Connection connection = ConnectionStorage.getConnection();
        PreparedStatement preparedStatement = connection.prepareStatement( sql: "INSERT INTO hot.prenotazione2(DataPrenotazione, DataArrivoCli
                "VALUES (?, ?, ?, ?, ?, ? , ?, ? , ? , ? , ?)");
        System.out.println(p);
        preparedStatement.setDate( parameterIndex: 1,Date.valueOf(p.getDataCreazionePrenotazione()));
        preparedStatement.setDate( parameterIndex: 2,Date.valueOf(p.getDataInizio()));
        preparedStatement.setDate( parameterIndex: 3,Date.valueOf(p.getDataFine()));
        preparedStatement.setString( parameterIndex: 4, p.getTrattamento().getNome());
        preparedStatement.setString( parameterIndex: 5,p.getNoteAggiuntive());
        preparedStatement.setString( parameterIndex: 6 , p.getIntestatario());
        preparedStatement.setDate( parameterIndex: 7, Date.valueOf(p.getDataScadenza()));
        preparedStatement.setString( parameterIndex: 8,p.getNumeroDocumento());
        preparedStatement.setDate( parameterIndex: 9,Date.valueOf(p.getDataRilascio()));
        preparedStatement.setString( parameterIndex: 10,p.getTipoDocumento());
        preparedStatement.setBoolean( parameterIndex: 11,p.getStatoPrenotazione());
        preparedStatement.setBoolean( parameterIndex: 12,p.isCheckIn());
        preparedStatement.executeUpdate();
```

```
@Test  Renato
@Tag("True")
@DisplayName("doSave() quando va tutto bene")
public void doSaveAllTrue() throws SQLException {
    prenotazioneDAO.doSave(prenotazione);
}
```

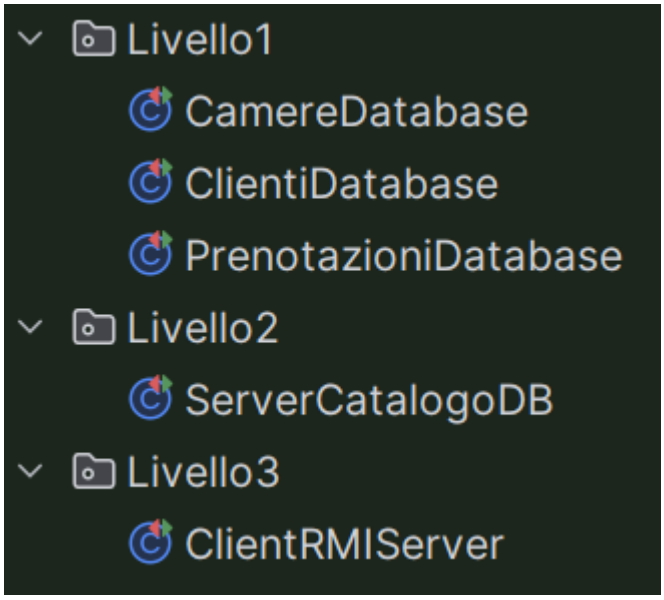
- **Test passato (Success):**

TEST RESULT INTEGRATION TESTS

Fare i test sia blackbox che whitebox è stato molto profittevole visto che abbiamo riscontrato molti meno errori nel test di integrazione e ci ha permesso di fare un test molto più agevole e veloce nell'integrazione. Abbiamo suddiviso il test Bottom Up in 3 livelli :

1. partendo dal DB con i cataloghi + classi DAO
2. logica server con chiamate tramite pattern command + observer al integrando il livello 1
3. logica Client con chiamate al registry RMI connesso al RMI server + chiamate al DB , risultando un vero e proprio test di sistema

NB essendo che il core del nostro progetto e i requisiti si concentrano più sul dare l'affidabilità al sistema RMI e garantire la trasparenza di locazione, non è abbiamo ritenuto rilevante testare l'interfaccia grafica.



ERRORE

TC4

Questo test , simula la chiamata al database tramite il catalogo clienti, l'operazione di retrieve dal Database. Grazie all'esecuzione del test , come risultato è stato constatato che il catalogo clienti no faceva correttamente l'update.

```
@Test  StefanoGit50
@DisplayName("Bottom up: simulazione della chiamata update tramite catalogo")
@Tag("integration")
public void updateCatalogoClienti() throws CloneNotSupportedException {
```

```
279                 preparedStatement.setInt( parameterIndex: 15,o.getCamera().getNumeroCamera
280                 preparedStatement.setDouble( parameterIndex: 16,o.getCamera().getPrezzoCam
281                 preparedStatement.setString( parameterIndex: 17, o.getCf());
282                 int rowsAffected = preparedStatement.executeUpdate();
283                 log.debug(rowsAffected+" rows affected");
284             }
285             finally
```

```
1 test failed, 3 passed  4 tests total, 2 sec 454 ms
"C:\Program Files\Java\jdk-25\bin\java.exe" ...
17:00:48 DEBUG CatalogoClienti: DEBUG: Sto per chiamare doUpdate sul DB!
17:00:48 DEBUG ClienteDAO: 0 rows affected
```

FIX

La soluzione è stata cambiare la query facendo in modo che nel database prenda le giuste informazioni e garantisca correttamente l'integrità dei dati

```
@Override
public synchronized void doUpdate(Cliente o) throws SQLException
{

    String query = "UPDATE cliente LEFT JOIN associato_a USING (CF) SET " +
        "nome = ?, cognome = ?, Cap = ?, comune = ?, " +
        "civico = ?, provincia = ?, via = ?, Email = ?, Sesso = ?, " +
        "telefono = ?, Nazionalita = ?, " +
        "DataDiNascita = ?, IsBackListed = ?, " +
        "NumeroCamera = ?, NumeroCameraStorico = ?, PrezzoAcquisto = ? " +
        "WHERE cliente.CF = ?";
    if(o != null && o.getCf() != null){
```

Test Case	Test Result
TC1	**FAIL**
TC2	**FAIL**
TC3	**FAIL**
TC4	**FAIL**
TC5	**PASS**
TC6	**PASS**
TC7	**PASS**
TC8	**PASS**
TC9	**PASS**
TC10	**PASS**
TC11	**PASS**
TC12	**PASS**
TC13	**PASS**
TC14	**PASS**
TC15	**PASS**
TC16	**PASS**
TC17	**PASS**
TC18	**PASS**
TC19	**PASS**
TC20	**PASS**
TC21	**PASS**
TC22	**PASS**
TC23	**PASS**
TC24	**PASS**

Test Case	Test Result
TC1	**FAIL**
TC2	**FAIL**
TC3	**FAIL**
TC4	**FAIL**
TC5	**PASS**
TC6	**PASS**
TC7	**PASS**
TC8	**PASS**
TC9	**PASS**
TC10	**PASS**
TC11	**PASS**
TC12	**PASS**
TC13	**PASS**
TC14	**PASS**
TC15	**PASS**
TC16	**PASS**
TC17	**PASS**
TC18	**PASS**
TC19	**PASS**
TC20	**PASS**
TC21	**PASS**
TC22	**PASS**
TC23	**PASS**
TC24	**PASS**