



UNIVERSITÀ DEGLI STUDI
DI SALERNO



TEST RESULT

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

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 documento di Test Result.	Giovanni Riccardi

Sommario

<i>TEST RESULT: BLACK BOX</i>	4
CATEGORY PARTITION	4
TEST RESULTS	4
TEST FAILUREs	5
PROPOSED FIX	9
TEST RESULTS – RE-RUN.....	10
<i>TEST RESULT: WHITE BOX</i>	11
TEST DI UNITÀ: WHITEBOX.....	24
TEST DI UNITÀ: WHITEBOX.....	11
TEST DI UNITÀ: WHITEBOX.....	15
TEST DI UNITÀ: WHITEBOX.....	17
TEST DI UNITÀ: WHITEBOX TC23	19
TEST DI UNITÀ: WHITEBOX TC25	22
Test Results prima.....	24
... e dopo.....	36
TEST WHITEBOX – COVERAGE.....	39
<i>TEST RESULT: INTEGRATION TESTS</i>	40
Test failures.....	40
FIX.....	41
Test results prima.....	42
... e dopo.....	43

NOTE:

La scelta di utilizzare RMI all'interno del sistema per la comunicazione tra il client ed il server ha avuto come conseguenza che tool come *Selenium* non sono adatti per il testing del sistema in questione. Questo, infatti, è stato incentrato più sul verificare l'affidabilità del sistema di chiamate tramite RMI e sul garantire la trasparenza di locazione; di conseguenza non è stato ritenuto rilevante testare l'interfaccia grafica, in quanto intercambiabile.

Il testing è stato svolto creando delle componenti che simulino le chiamate ai metodi dell'interfaccia remota, simulando quindi il comportamento del client interponendosi tra il *presentation layer* e l'*application layer*.

Ogni test (in essenza) si svolge nel seguente:

1. Una componente crea i parametri necessari per chiamare il metodo remoto da testare;
 - a. le *pre-condizioni* sui parametri in ingresso possono essere rispettate o meno, in funzione che lo scopo del test sia proprio quello di verificare il comportamento del metodo in condizioni di uno o più input errati;
2. Si verifica che l'output sia conforme con le *post-condizioni*.

TEST RESULT: BLACK BOX

CATEGORY PARTITION

Si vuole testare la seguente funzionalità: "Registrazione di una registrazione". Per avviare i test si usa la seguente test suite:

```

8  @Suite & RccGnn
9  @SuiteDisplayName("Test suite: registrazione nuova prenotazione!")
10 @SelectPackages({"blackbox.RegistraPrenotazione"})
11 @IncludeTags({"registraPrenotazione"})
12 public class TestSuiteRegistraPrenotazione {
13 }
14

```

1.0: Codice test suite

TEST RESULTS

La seguente tabella mostra gli esiti dell'esecuzione della test suite col sistema nello stato attuale.

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 17
- TC1: riga 192
- TC1: riga 217
- TC1: riga 241

Di seguito si mostrano i punti in cui sono stati lanciati gli errori nelle immagini [1.1], [1.2], [1.3], [1.4]:

```

165     @Test @RccGnn
166     @DisplayName("TC1: [Success] Registrazione con servizi: nessuno")
167     void testCase1() throws RemoteException{
168         Prenotazione p = createBasePrenotazione(), campione;
169         p.setIDPrenotazione(autoIncrement);
170         p.setTrattamento(null);
171         p.setPrezzoAcquistoTrattamento(0.0);
172         p.setListaServizi(null);
173
174         Assertions.assertDoesNotThrow(() -> frontDesk.addPrenotazione(p));
175         try {
176             campione = frontDesk.getPrenotazioneById(autoIncrement);
177             autoIncrement++;
178             Assertions.assertEquals(p, campione);
179         } catch (RemoteException e) {
180             e.printStackTrace();
181         }
182     }

```

1.1: errore TC1, il punto in cui è stato lanciato l'errore è evidenziato

```

185     @Test @RccGnn
186     @DisplayName("TC2: [Success] Registrazione con servizi: 1")
187     void testCase2() {
188         Prenotazione p = createBasePrenotazione(), campione;
189         ArrayList<Servizio> c = new ArrayList<>();
190         c.add(listaServizi.getFirst());
191         p.setListaServizi(c);
192         p.setIDPrenotazione(autoIncrement);
193         Assertions.assertDoesNotThrow(() -> frontDesk.addPrenotazione(p));
194
195         try {
196             campione = frontDesk.getPrenotazioneById(autoIncrement);
197             autoIncrement++;
198             Assertions.assertEquals(p, campione);
199         } catch (RemoteException e) {
200             e.printStackTrace();
201         }
202     }

```

1.2: errore TC2, il punto in cui è stato lanciato l'errore è evidenziato

```

203     @Test @RccGnn
204     @DisplayName("TC3: [Success] Registrazione con servizi: 3 e tipo documento: CID")
205     void testCase3() {
206         Prenotazione p = createBasePrenotazione(), campione;
207         p.setTipoDocumento("CID");
208         p.setTrattamento(new Trattamento( nome: "Pensione Completa", prezzo: 55.0));
209         p.setPrezzoAcquistoTrattamento(55.0);
210         ArrayList<Cliente> c = new ArrayList<>();
211         c.add(listaClienti.getFirst());
212         p.setListaClienti(c);
213         c.add(listaClienti.getFirst());
214         p.setIntestatario(c.getFirst().getNome() + " " + c.getFirst().getCognome());
215         p.setIDPrenotazione(autoIncrement);
216
217         Assertions.assertDoesNotThrow(() -> frontDesk.addPrenotazione(p));
218         try {
219             campione = frontDesk.getPrenotazioneById(autoIncrement);
220             autoIncrement++;
221             Assertions.assertEquals(p, campione);
222         } catch (RemoteException e) {
223             e.printStackTrace();
224         }
225     }

```

1.3: errore TC3, il punto in cui è stato lanciato l'errore è evidenziato

```

227     @Test & RccGnn *
228     @DisplayName("TC4: [Success] Clienti multipli (2) e camere multiple (2) con tipo documento: Passaporto")
229     void testCase4() {
230         Prenotazione p = createBasePrenotazione(), campione;
231         p.setTipoDocumento("Passaporto");
232         p.setTrattamento(new Trattamento( nome: "Solo pernottamento", prezzo: 0));
233         p.setPrezzoAcquistoTrattamento(0.0);
234         ArrayList<Cliente> c = new ArrayList<>();
235         c.add(listaClienti.get(1));
236         c.add(listaClienti.getFirst()); // 2 clienti -> 2 camere diverse
237         p.setIntestatario(c.getFirst().getNome() + " " + c.getFirst().getCognome());
238         p.setListaClienti(c);
239         p.setIdPrenotazione(autoIncrement);
240
241         Assertions.assertDoesNotThrow(() -> frontDesk.addPrenotazione(p));
242         try {
243             campione = frontDesk.getPrenotazioneById(autoIncrement);
244             Assertions.assertEquals(p, campione);
245             autoIncrement++;
246         } catch (RemoteException e) {
247             Assertions.fail(e.getMessage());
248         }
249     }

```

1.4: errore TC4, il punto in cui è stato lanciato l'errore è evidenziato

L'errore lanciato è lo stesso per tutti e quattro i test case, così come rappresentato nelle figure [1.5], [1.6], [1.7], [1.8].

```

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:174) <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.checkNotNull(CatalogueUtils.java:37)
    at it.unisa.Server.gestionePrenotazioni.FrontDesk.addPrenotazione(FrontDesk.java:130)
>     <6 internal lines>
><13 folded frames>

```

1.5: descrizione dell'errore per il TC1

```

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.checkNotNull(CatalogueUtils.java:37)
    at it.unisa.Server.gestionePrenotazioni.FrontDesk.addPrenotazione(FrontDesk.java:130)
>     <6 internal lines>
><13 folded frames>

```

1.6: descrizione dell'errore per il TC2

```

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:217) <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.checkNotNull(CatalogueUtils.java:37)
    at it.unisa.Server.gestionePrenotazioni.FrontDesk.addPrenotazione(FrontDesk.java:130)
>     <6 internal lines>
><13 folded frames>

```

1.7: descrizione dell'errore per il TC3

```

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:241) <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.checkNotNull(CatalogueUtils.java:37)
    at it.unisa.Server.gestionePrenotazioni.FrontDesk.addPrenotazione(FrontDesk.java:130)
>     <6 internal lines>
><13 folded frames>

```

1.8: descrizione dell'errore per il TC4

L'errore lanciato è chiama *InvalidInputException* (un'eccezione non controllata definita dal

sistema) ed è causato dal fatto che il sistema, nello stato attuale, rileva quando un campo usato per registrare una prenotazione è vuoto o nullo ed impedisce la registrazione della prenotazione sul database.

Il metodo che lancia l'errore è il metodo *checkNull(Prenotazione)*, chiamato nel seguente punto:

```
126     // COMANDI PRENOTAZIONE
127     @Override 32 usages & RccGnn
128     public void addPrenotazione(Prenotazione p) throws RemoteException {
129         CatalogueUtils.checkNotNull(p);          // Lancia InvalidInputException
130         CatalogoPrenotazioni.checkPrenotazione(p); // Lancia InvalidInputException
131         AddPrenotazioneCommand command = new AddPrenotazioneCommand(p);
132         invoker.executeCommand(command);
133     }
```

1.9: metodo che chiama *checkNull()*, il quale lancia l'errore

Analizzando il metodo si nota che questo lancia un'eccezione nel caso in cui una qualsiasi delle variabili di istanza di un oggetto prenotazione è nullo (includendo stringhe vuote o blank come "" o " " e liste vuote), quindi includendo "trattamento" o "metodo di pagamento". Questo non si allinea con quelli che sono i requisiti, in quanto è possibile avere una prenotazione in cui alcuni campi sono nulli (ad esempio, si può prenotare senza scegliere un trattamento, come si può anche non scegliere un servizio).

PROPOSED FIX

Dai requisiti del progetto, si inferisce che le seguenti informazioni possono non essere dichiarate al momento in cui si istanzia una prenotazione:

- *note aggiuntive* → non si aggiunge alcuna descrizione aggiuntiva alla prenotazione;
- *trattamento* → è possibile prenotare senza trattamento;
- *prezzo acquisto trattamento* → inerentemente al precedente, se non si sceglie alcun trattamento;
- metodo di pagamento e data emissione della ricevuta fiscale → sono rilasciate al check-out.

Una possibilità è quella di far saltare il controllo di questi campi al metodo, in modo tale da poter evitare il lancio dell'eccezione.

Di seguito si riportano le modifiche al metodo proposte:

```
18 @... public static <T> void checkNull(T oggetto) throws InvalidInputException { 2 usages & RccGnn*  
19     Class<?> oggettoClass = oggetto.getClass();  
20     Field[] fields = oggettoClass.getDeclaredFields(); // Ottieni tutti i campi della classe di appartenenza dell'oggetto  
21     List<String> excludedFields = List.of(  
22         "noteaggiuntive", "datascadenzatoken", "trattamento", "metododipagamento", "dataemissionericevuta",  
23         "prezzoacquistotrattamento"  
24     );  
25     for (Field field : fields) {  
26         try {  
27             if (excludedFields.contains(field.getName().toLowerCase())) {  
28                 field.setAccessible(false);  
29                 continue;  
30             }  
31             field.setAccessible(true); // Rendili accessibili temporaneamente  
32             Object fieldValue = field.get(oggetto);  
33             if (fieldValue == null) {  
34                 throw new InvalidInputException("Campo: " + field.getName() + " è nullo");  
35             }  
36             if (field.getType().equals(String.class) && ((String) fieldValue).isBlank()) {  
37                 throw new InvalidInputException("Campo: " + field.getName() + " è vuoto");  
38             }  
39             field.setAccessible(false); // Rendili di nuovo non accessibili...  
40         } catch (IllegalAccessException e) {  
41             field.setAccessible(false); // ...Anche in caso di errore  
42             e.printStackTrace();  
43         }  
44     }  
}
```

1.10: metodo *checkNull()* corretto

TEST RESULTS – RE-RUN

A questo punto si esegue nuovamente la test suite.

Test Case	Test Result
TC1	PASS
TC2	PASS
TC3	PASS
TC4	PASS
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 TC1

Titolo: Test salvataggio multiplo

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.

```
@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"));
}
```

1.1: Test suite

```
java.sql.SQLSyntaxErrorException: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the
                                              ^

at com.mysql.cj.jdbc.exceptions.SQLException.createSQLException(SQLException.java:112)
at com.mysql.cj.jdbc.exceptions.SQLExceptionsMapping.translateException(SQLExceptionsMapping.java:114)
at com.mysql.cj.jdbc.ClientPreparedStatement.executeInternal(ClientPreparedStatement.java:988)
at com.mysql.cj.jdbc.ClientPreparedStatement.executeUpdateInternal(ClientPreparedStatement.java:1166)
at com.mysql.cj.jdbc.ClientPreparedStatement.executeUpdateInternal(ClientPreparedStatement.java:1101)
at com.mysql.cj.jdbc.ClientPreparedStatement.executeLargeUpdate(ClientPreparedStatement.java:1448)
at com.mysql.cj.jdbc.ClientPreparedStatement.executeUpdate(ClientPreparedStatement.java:1084)
at it.unisa.Storage.DAO.CameraDAO.doSaveAll(CameraDAO.java:130)
```

1.2: Codice errato

```

@Override
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(",");
        }
    }

    // Riempì 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);
        }
    }
}

```

1.3: Errore riscontrato (Test Failure)

```

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(",");
            }
        }

        // Riempì 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");
    }
}

```

1.4: Correzione applicata al metodo che lanciava l'eccezione

```

✓ 1 test passed 1 test total, 817 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

```

1.5: Test case trattato che passa con esito positivo

TEST DI UNITÀ: WHITEBOX TC12

Titolo: Test di recupero cliente

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.

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

2.1: Test case

```
org.opentest4j.AssertionFailedError: Unexpected exception thrown: java.lang.ClassCastException: class java.lang.String cannot be cast to class
> <5 internal lines>
>   at WhiteBox.UnitTest.ClienteDAOTesting.doRetrieveByKey(ClienteDAOTesting.java:48) <1 internal line>
  at java.base/java.util.ArrayList.forEach(ArrayList.java:1604)
<1 folded frame>
Caused by: java.lang.ClassCastException Create breakpoint : class java.lang.String cannot be cast to class java.sql.Date (java.lang.String is in mo
  at it.unisa.Storage.DAO.ClienteDAO.doRetrieveByKey(ClienteDAO.java:123)
>   at WhiteBox.UnitTest.ClienteDAOTesting.lambda$doRetrieveByKey$0(ClienteDAOTesting.java:48) <1 internal line>
... 6 more
```

2.2: Errore riscontrato (Test Failure)

```

@Override & Renato Natale +2*
public synchronized Cliente doRetrieveByKey(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,
        Integer civico = null;
        LocalDate date = null;
        Boolean isBlacklisted = false;
        String nazionalita = 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( columnIndex: 1);
                nome = (String) resultSet.getObject( columnIndex: 2);
                cognome = (String) resultSet.getObject( columnIndex: 3);
                cap = (String) resultSet.getObject( columnIndex: 4);
                comune = (String) resultSet.getObject( columnIndex: 5);
                civico = (Integer) resultSet.getObject( columnIndex: 6);
                provincia = (String) resultSet.getObject( columnIndex: 7);
                via = (String) resultSet.getObject( columnIndex: 8);
                email = (String) resultSet.getObject( columnIndex: 9);
                sesso = (String) resultSet.getObject( columnIndex: 10);
                telefono = (String) resultSet.getObject( columnIndex: 11);
                cittadinanza = (String) resultSet.getObject( columnIndex: 12);
                Date date1 = (Date) resultSet.getObject( columnIndex: 13);
                date = date1.toLocalDate();
                isBlacklisted = (Boolean) resultSet.getObject( columnIndex: 14);
                nazionalita = resultSet.getString( columnLabel: "Nazionalità");
            }
            resultSet.close();
        }finally{
            if(con != null){
                ConnectionStorage.releaseConnection(con);
            }
        }
    }
}

```

2.3: Codice causa dell'errore (Bugged Code)

```

@Override & Renato Natale +2*
public synchronized Cliente doRetrieveByKey(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 nazionalita = 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: "DataDiNascita");
                date = date1.toLocalDate();
                isBlacklisted = (Boolean) resultSet.getObject( columnLabel: "IsBlacklisted");
                nazionalita = resultSet.getString( columnLabel: "Nazionalità");
            }
            resultSet.close();
        }finally{
    }
}

```

2.4: Fix applicato (Corrected Code), sul metodo di test

```
✓ 1 test passed 1 test total, 1 sec 391 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
```

2.5: Test passato (Success):

TEST DI UNITÀ: WHITEBOX TC17

Titolo: Test di recupero multiplo clienti 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.

```
java.lang.ClassCastException: class java.lang.String cannot be cast to class java.sql.Date (java.lang.String is in module java.base of loader
 at it.unisa.Storage.DAO.ClienteDAO.doRetrieveAll(ClienteDAO.java:209)
 at WhiteBox.UnitTest.ClienteDAOTesting.doRetrieveAll(ClienteDAOTesting.java:89) <1 internal line>
 at java.base/java.util.ArrayList.forEach(ArrayList.java:1604)
 at java.base/java.util.ArrayList.forEach(ArrayList.java:1604)
```

3.1: Descrizione dell'errore

```

@Override & Renato Natale +2 *
public synchronized Collection<Cliente> doRetrieveAll(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 nazionalita = resultSet.getString( columnLabel: "Nazionalita");
            cliente = new Cliente(nome,cognome,cittadinazione,provincia,comune,via,civico, Integer.parseInt(cap),telefono,sesso,date,cf1,email,nazionalita);
            cliente.setBlacklisted(isBackListed);
            clientes.add(cliente);
        }
    }
}

```

3.2: Codice causa dell'errore (Bugged Code):

```

@Override & Renato Natale +2 *
public synchronized Collection<Cliente> doRetrieveAll(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 nazionalita = resultSet.getString( columnLabel: "Nazionalità");
            cliente = new Cliente(nome,cognome,cittadinazione,provincia,comune,via,civico, Integer.parseInt(cap),telefono,sesso,date,cf1,email,nazionalita);
            cliente.setBlacklisted(isBackListed);
            clientes.add(cliente);
        }
    }
}

```

3.3: Fix applicato

```

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

    // clientes1.add(new Cliente("Mario", "Rossi", "Italiana", "Napoli", "Napoli", "Via Roma", 15, 80100, "3331234567", "Maschio", LocalDate.of(1990, 1, 1));
    // clientes1.add(new Cliente("Laura", "Verdi", "Italiana", "Roma", "Roma", "Via Milano", 23, 100, "3339876543", "Femmina", LocalDate.of(1990, 1, 1));

    assertEquals(clientes1, clientes);
}

```

3.4: Metodo di test

```

✓ 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

```

3.4: Test success

TEST DI UNITÀ: WHITEBOX TC36

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 l'oggetto che doveva essere rappresentato come valore di superamento del test 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.

```

org.opentest4j.AssertionFailedError:
Expected :[Cliente{nome='Mario', cf='RSSMRA85M01HS01Z', email='mario.rossi@email.it', cognome='Rossi', cittadinanza='Italiana', provincia='Napo
Actual   :[Cliente{nome='Mario', cf='RSSMRA85M01HS01Z', email='mario.rossi@email.it', cognome='Rossi', cittadinanza='Italiana', provincia='Napo
<Click to see difference>

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

```

4.1: Errore riscontrato (Test Failure)

```

@Test & Renato *
@Tag("True")
@DisplayName("doRetrieveByAttribute() quando va tutto bene")
public void doRetrieveByAttributeAllTrue() throws SQLException {
    ArrayList<Cliente> clientes = new ArrayList<>();
    // clientes.add(new Cliente("Mario","Rossi","Italiana","Napoli","Napoli","via Roma",15,80100,"3331234567","Maschio",LocalDate.of(1996,1,1));
    // clientes.add(new Cliente("Laura","Verdi","Italiana","Roma","Roma","Via Milano",23,100,"3339876543","Femmina",LocalDate.of(1996,1,1));
    Object s = "Italiana";
    ArrayList<Cliente> clientes1 = (ArrayList<Cliente>) clienteDAO.doRetrieveByAttribute(attribute: "Cittadinanza",s);
    assertEquals(clientes,clientes1);
}

```

4.2: Metodo di test

```

@Override & Renato *
public synchronized Collection<Cliente> doRetrieveByAttribute(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());
                lista.add(cliente);
            }
        }finally{
            if(con!= null){
                if (preparedStatement != null) {
                    preparedStatement.close();
                }
            }
        }
    }
}

```

4.3: Metodo che causa l'errore

```

@Override & RccGnn +2 *
public synchronized Collection<Cliente> doRetrieveByAttribute(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);
            }
        }
    }
}

```

4.4: Fix applicato (Corrected Code)

```

✓ 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

```

4.5: Test success

TEST DI UNITÀ: WHITEBOX TC25

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.

```
com.mysql.cj.jdbc.exceptions.MySQLDataTruncation: Data truncation: Incorrect date value: 'AA123456' for column 'dataScadenza' at row 1
    at com.mysql.cj.jdbc.exceptions.SQLExceptionsMapping.translateException(SQLExceptionsMapping.java:96)
    at com.mysql.cj.jdbc.ClientPreparedStatement.executeInternal(ClientPreparedStatement.java:988)
    at com.mysql.cj.jdbc.ClientPreparedStatement.executeUpdateInternal(ClientPreparedStatement.java:1166)
    at com.mysql.cj.jdbc.ClientPreparedStatement.executeUpdateInternal(ClientPreparedStatement.java:1101)
    at com.mysql.cj.jdbc.ClientPreparedStatement.executeLargeUpdate(ClientPreparedStatement.java:1448)
    at com.mysql.cj.jdbc.ClientPreparedStatement.executeUpdate(ClientPreparedStatement.java:1084)
    at it.unisa.Storage.DAO.PrenotazioneDAO.doSave(PrenotazioneDAO.java:57)
    at WhiteBox.UnitTest.prenotazioneDAOTesting.doSaveAllTrue(prenotazioneDAOTesting.java:50)
<3 folded frames>
```

5.1: *Errore riscontrato (Test Failure):*

```

@Override & Renato Natale +3
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 (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)");
        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();

        // Salva il trattamento associato
        if(p.getTrattamento().getNome() != null){
            String query = "UPDATE hot.trattamento2 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();
            }
        }
    }
}

```

5.2: Codice causa dell'errore (Bugged Code):

```

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();
    }
}

```

5.3: Fix applicato (Corrected Code)

```

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

```

5.4: Metodo di testing

TEST DI UNITÀ: WHITEBOX TC25

Titolo: Test di salvataggio prenotazione

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".

```

@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));
}

```

6.1:

Metodo di testing

```

.org.junit.AssertionFailedError: Unexpected exception thrown: java.lang.NullPointerException: Cannot invoke "java.sql.PreparedStatement.setInt(int, int)" because "stmt" is null
internal lines>
at WhiteBox.UnitTest.prenotazioneDAOTesting.doSaveAllFalse(prenotazioneDAOTesting.java:157) <1 internal line>
at java.base/java.util.ArrayList.forEach(ArrayList.java:1694)
... 6 more
Caused by: java.lang.NullPointerException Create breakpoint : Cannot invoke "java.sql.PreparedStatement.setInt(int, int)" because "stmt" is null
at it.unisa.Storage.DAO.PrenotazioneDAO.doSave(PrenotazioneDAO.java:63)
at WhiteBox.UnitTest.prenotazioneDAOTesting.lambda$doSaveAllFalse$0(prenotazioneDAOTesting.java:157) <1 internal line>
... 6 more

27, 2026 1:41:25 PM org.junit.platform.launcher.core.DiscoveryEngine$IssueNotifier logIssues
WARNING: TestEngine with ID 'junit-jupiter' encountered 2 non-critical issues during test discovery:

[WARNING] Invalid tag syntax in @Tag("") declaration on method 'public void WhiteBox.UnitTest.prenotazioneDAOTesting.doRetrieveByKeyAllTrue() throws java.sql.SQLException'. Tag will be ignored.
Source: MethodSource [className = 'WhiteBox.UnitTest.prenotazioneDAOTesting', methodName = 'doRetrieveByKeyAllTrue', methodParameterTypes = '']
at WhiteBox.UnitTest.prenotazioneDAOTesting.doRetrieveByKeyAllTrue(SourceFile:8)

[WARNING] @DisplayName on method 'public void WhiteBox.UnitTest.prenotazioneDAOTesting.doRetrieveByKeyAllTrue() throws java.sql.SQLException' must be declared with a non-blank value.
Source: MethodSource [className = 'WhiteBox.UnitTest.prenotazioneDAOTesting', methodName = 'doRetrieveByKeyAllTrue', methodParameterTypes = '']
at WhiteBox.UnitTest.prenotazioneDAOTesting.doRetrieveByKeyAllTrue(SourceFile:8)

27, 2026 1:41:25 PM it.unisa.Storage.ConnectionStorage shutdown
: Chiusura connessioni...

```

6.2: Test failure

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

6.3: Linea di codice che causa l'errore

```
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();  
        }  
    }  
}
```

6.4: Correzione applicata

```
✓ 1 test passed 1 test total, 1 sec 614 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  
gen 27, 2026 2:56:15 PM org.junit.platform.launcher.core.DiscoveryIssueNotifier logIssues  
WARNING: TestEngine with ID 'junit-jupiter' encountered 2 non-critical issues during test discovery:  
  
(1) [WARNING] Invalid tag syntax in @Tag("") declaration on method 'public void WhiteBox.UnitTest.prenotazioneDAOTesting.doRetrieveByKeyAllTrue()  
Source: MethodSource [className = 'WhiteBox.UnitTest.prenotazioneDAOTesting', methodName = 'doRetrieveByKeyAllTrue', methodParameterTypes =  
at WhiteBox.UnitTest.prenotazioneDAOTesting.doRetrieveByKeyAllTrue(SourceFile:0)  
  
(2) [WARNING] @DisplayName on method 'public void WhiteBox.UnitTest.prenotazioneDAOTesting.doRetrieveByKeyAllTrue() throws java.sql.SQLException'  
Source: MethodSource [className = 'WhiteBox.UnitTest.prenotazioneDAOTesting', methodName = 'doRetrieveByKeyAllTrue', methodParameterTypes =  
at WhiteBox.UnitTest.prenotazioneDAOTesting.doRetrieveByKeyAllTrue(SourceFile:0)  
gen 27, 2026 2:56:15 PM it.unisa.Storage.ConnectionStorage shutdown  
INFO: Chiusura connessioni...
```

6.5: Test passato (success)

TEST DI UNITÀ: WHITEBOX TC28

Titolo: Test di recupero di tutte le prenotazioni presenti nel sistema.

Si testa se il sistema è in grado di recuperare tutte le prenotazioni presenti nel sistema. Il test è stato strutturato in modo tale che ogni campo di ciascuna prenotazione recuperata sia diverso da *null* (anche quelli che possono esserlo).

Descrizione dell'errore:

L'errore è lanciato all'ultima istruzione quando di esegue *AssertEquals()* sulle liste di prenotazioni. Questo è dovuto al campo che memorizza la data di Emissione della ricevuta fiscale, che è *null*. L'errore è dovuto ad una mancata lettura del campo dal Database da parte del metodo delegato al retrieve *doRetrieveByKey()*.

Fix:

La soluzione è semplicemente quella di includere la lettura da database del campo menzionato prima.

```
org.opentest4j.AssertionFailedError:  
Expected :[Prenotazione{IDPrenotazione=1, cittadinanza='italiana', metodoDiPagamento='Carta di Credito', dataCreazionePrenotazione=2023-12-01, dataInizio=  
Actual   :[Prenotazione{IDPrenotazione=1, cittadinanza='italiana', metodoDiPagamento='Carta di Credito', dataCreazionePrenotazione=2023-12-01, dataInizio=  
<Click to see difference>
```

7.1: errore lanciato

```

public class PrenotazioneDAOTesting {
    public void doRetrieveAllAllTrue() throws SQLException {
        //prenotazioneDAO.doDelete();
        ArrayList<Prenotazione> prenotazioni = (ArrayList<Prenotazione>) prenotazioneDAO.doRetrieveAll( order: "IDPrenotazione");
        ArrayList<Prenotazione> prenotazioni1 = new ArrayList<>();
        ArrayList<Cliente> clienti = new ArrayList<>();
        ArrayList<Servizio> servizi = new ArrayList<>();
        System.out.println(prenotazioni);
        Clientes.add(new Cliente( nome: "Hans", cognome: "Muller", provincia: "Berlin", comune: "Berlino", via: "Alexanderplatz", numeroCivico: 10, CA
        servizi.add(new Servizio( nome: "Transfer Aeroporto", prezzo: 35.0));
        Prenotazione prenotazione1 = new Prenotazione(LocalDate.of( year: 2024, month: 2, dayOfMonth: 15),LocalDate.of( year: 2024, month: 3, dayOfMonth: 15));
        prenotazione1.setIDPrenotazione(3);

        servizi = new ArrayList<>();
        Clientes = new ArrayList<>();
        Clientes.add(new Cliente( nome: "Luigi", cognome: "Verdi", provincia: "Milano", comune: "Milano", via: "Corso Buenos Aires", numeroCivico: 10, CA
        servizi.add(new Servizio( nome: "Spa e Benessere", prezzo: 45.0));
        servizi.add(new Servizio( nome: "Spa e Benessere", prezzo: 45.0));
        servizi.add(new Servizio( nome: "Colazione in Camera", prezzo: 12.0));
        servizi.add(new Servizio( nome: "Colazione in Camera", prezzo: 12.0));
        Prenotazione prenotazione2 = new Prenotazione(LocalDate.of( year: 2024, month: 2, dayOfMonth: 1),LocalDate.of( year: 2024, month: 2, dayOfMonth: 1));
        prenotazione2.setIDPrenotazione(2);
        prenotazione2.setCheckIn(true);

        Clientes = new ArrayList<>();
        Clientes.add(new Cliente( nome: "Mario", cognome: "Rossi", provincia: "Roma", comune: "Roma", via: "Via del Corso", numeroCivico: 10, CA
        servizi = new ArrayList<>();
        servizi.add(new Servizio( nome: "Parcheggio", prezzo: 10.0));
        servizi.add(new Servizio( nome: "Parcheggio", prezzo: 10.0));
        Prenotazione prenotazione3 = new Prenotazione(LocalDate.of( year: 2023, month: 12, dayOfMonth: 1),LocalDate.of( year: 2024, month: 1, dayOfMonth: 1));
        prenotazione3.setIDPrenotazione(1);

        prenotazioni1.add(prenotazione3);
        prenotazioni1.add(prenotazione2);
        prenotazioni1.add(prenotazione1);
        assertEquals(prenotazioni1,prenotazioni);
    }
}

```

7.2: metodo di testing che ha trovato l'errore

```

p.setIDPrenotazione(key);
p.setTrattamento(t);
p.setPrezzoAcquistoTrattamento(prezzoTratt);
p.setListaClienti(clienti);
p.setIntestatario(rs.getString( columnLabel: "NomeIntestatario"));
p.setDataCreazionePrenotazione(rs.getDate( columnLabel: "DataCreazionePrenotazione").toLocalDate());
p.setDataFine(rs.getDate( columnLabel: "DataPartenzaCliente").toLocalDate());
p.setDataInizio(rs.getDate( columnLabel: "DataArrivoCliente").toLocalDate());
p.setDataRilascio(rs.getDate( columnLabel: "DataRilascioDocumento").toLocalDate());
p.setDataScadenza(rs.getDate( columnLabel: "DataScadenzaDocumento").toLocalDate());
p.setStatoPrenotazione(rs.getBoolean( columnLabel: "Stato"));
p.setCheckIn(rs.getBoolean( columnLabel: "CheckIn"));
p.setNoteAggiuntive(rs.getString( columnLabel: "NoteAggiuntive"));
p.setNumeroDocumento(rs.getString( columnLabel: "NumeroDocumento"));
p.setMetodoPagamento(rs.getString( columnLabel: "MetodoPagamento"));
p.setCittadinanza(rs.getString( columnLabel: "Cittadinanza"));
p.setTipoDocumento(rs.getString( columnLabel: "TipoDocumento"));

```

7.3: codice errato

```

p.setIDPrenotazione(key);
p.setTrattamento(t);
p.setPrezzoAcquistoTrattamento(prezzoTrattamento);
p.setListaClienti(clienti);
p.setIntestatario(rs.getString(columnLabel: "NomeIntestatario"));
p.setDataCreazionePrenotazione(rs.getDate(columnLabel: "DataCreazionePrenotazione").toLocalDate());
p.setDataFine(rs.getDate(columnLabel: "DataPartenzaCliente").toLocalDate());
p.setDataInizio(rs.getDate(columnLabel: "DataArrivoCliente").toLocalDate());
p.setDataRilascio(rs.getDate(columnLabel: "DataRilascioDocumento").toLocalDate());
p.setDataScadenza(rs.getDate(columnLabel: "DataScadenzaDocumento").toLocalDate());
p.setDataEmissioneRicevuta(rs.getDate(columnLabel: "DataEmissioneRicevuta").toLocalDate());
p.setStatoPrenotazione(rs.getBoolean(columnLabel: "Stato"));
p.setCheckIn(rs.getBoolean(columnLabel: "CheckIn"));
p.setNoteAggiuntive(rs.getString(columnLabel: "NoteAggiuntive"));
p.setNumeroDocumento(rs.getString(columnLabel: "NumeroDocumento"));
p.setMetodoPagamento(rs.getString(columnLabel: "MetodoPagamento"));
p.setCittadinanza(rs.getString(columnLabel: "Cittadinanza"));
p.setTipoDocumento(rs.getString(columnLabel: "TipoDocumento"));

```

7.4: correzione proposta

TEST DI UNITÀ: WHITEBOX TC32

Titolo: Test di recupero di una prenotazione.

Si prova a recuperare una prenotazione dal database.

Descrizione dell'errore:

L'errore lanciato è un *NullPointerException*, lanciato alla penultima istruzione quando si esegue *AssertNotThrow()*. Questa viene lanciata perché il campo trattamento di prenotazione è *null* e, nel metodo delegato ad effettuare il salvataggio delle modifiche alla prenotazione, viene invocato il metodo *getNome()* sul trattamento (non verifica se questo sia stato modificato o meno).

Fix:

Basta tener conto della possibilità che il trattamento sia *null* ed impostare i campi della prenotazione in maniera adeguata (" " e 0 rispettivamente per i campi *nome* e *prezzoTrattamento*).

```

@Test new *
@Tags({@Tag("Exception"), @Tag("Error")})
@DisplayName("TC32: doRetrieveByKey() quando non trova il cliente")
public void doRetrieveByKeyIlSecondoResultSetDaException() throws SQLException {
    prenotazione.setTrattamento(null);
    assertDoesNotThrow(() -> prenotazioneDAO.doUpdate(prenotazione));
    assertThrows(SQLException.class, () -> prenotazioneDAO.doRetrieveByKey(prenotazione.getIDPrenotazione));
}

```

9.1: metodo di testing

```

org.opentest4j.AssertionFailedError: Unexpected exception thrown: java.lang.NullPointerException: Cannot invoke "it.unisa.Common.Trattamento.getNome()" because the return value of "it.unisa.Storage.DAO.PrenotazioneDAO.doUpdate(PrenotazioneDAO.java:441)" is null
<5 internal lines>
    at WhiteBox.UnitTest.prenotazioneDAOTesting.doRetriveByKeyIlSecondoResultSetDaException(prenotazioneDAOTesting.java:137) <1 internal line>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1604)
<1 folded frame>
Caused by: java.lang.NullPointerException Create breakpoint : Cannot invoke "it.unisa.Common.Trattamento.getNome()" because the return value of "it.unisa.Storage.DAO.PrenotazioneDAO.doUpdate(PrenotazioneDAO.java:441)" is null
    at WhiteBox.UnitTest.prenotazioneDAOTesting.lambda$doRetriveByKeyIlSecondoResultSetDaException$0(prenotazioneDAOTesting.java:137) <1 internal line>
    ... 6 more

```

9.2: errore lanciato

```

        attributi[3] = p.getDataInizio().toString();
        preparedStatement.setDate( parameterIndex: 5, Date.valueOf(p.getDataFine()));
        attributi[4] = p.getDataFine().toString();
        preparedStatement.setDate( parameterIndex: 6, p.getDataEmissioneRicevuta() != null ?
        attributi[5] = p.getDataEmissioneRicevuta() != null ? p.getDataEmissioneRicevuta()
        preparedStatement.setString( parameterIndex: 7, p.getNumeroDocumento());
        attributi[6] = p.getNumeroDocumento();
        preparedStatement.setDate( parameterIndex: 8, Date.valueOf(p.getDataRilascio()));
        attributi[7] = p.getDataRilascio().toString();
        preparedStatement.setDate( parameterIndex: 9, Date.valueOf(p.getDataScadenza()));
        attributi[8] = p.getDataScadenza().toString();

        if(p.getTrattamento()!=null){
            preparedStatement.setString( parameterIndex: 10, p.getTrattamento().getNome().trim());
            attributi[9] = p.getTrattamento().getNome().trim();
        }else{
            preparedStatement.setNull( parameterIndex: 10, Types.VARCHAR);
            attributi[9] = "";
        }

        preparedStatement.setString( parameterIndex: 11, p.getNoteAggiuntive().trim());
        attributi[10] = p.getNoteAggiuntive().trim();
        preparedStatement.setString( parameterIndex: 12, p.getTipoDocumento().trim());
        attributi[11] = p.getTipoDocumento().trim();
        preparedStatement.setBoolean( parameterIndex: 13, p.getStatoPrenotazione());
        attributi[12] = String.valueOf(p.getStatoPrenotazione());
        preparedStatement.setBoolean( parameterIndex: 14, p.isCheckIn());
        attributi[13] = String.valueOf(p.isCheckIn());
        preparedStatement.setDouble( parameterIndex: 15, p.getTrattamento().getPrezzo());
        attributi[14] = String.valueOf(p.getTrattamento().getPrezzo());
        preparedStatement.setString( parameterIndex: 16, p.getMetodoPagamento());
        attributi[15] = String.valueOf(p.getMetodoPagamento());
        preparedStatement.setString( parameterIndex: 17, p.getCittadinanza());
        attributi[16] = String.valueOf(p.getCittadinanza());
        preparedStatement.setInt( parameterIndex: 18,p.getIDPrenotazione());

```

9.3: Codice errato

```

attributi[6] = p.getNumeroDocumento();
preparedStatement.setDate( parameterIndex: 8, Date.valueOf(p.getDataRilascio()));
attributi[7] = p.getDataRilascio().toString();
preparedStatement.setDate( parameterIndex: 9, Date.valueOf(p.getDataScadenza()));
attributi[8] = p.getDataScadenza().toString();

if(p.getTrattamento()!=null){
    preparedStatement.setString( parameterIndex: 10, p.getTrattamento().getNome().trim());
    attributi[9] = p.getTrattamento().getNome().trim();
    preparedStatement.setDouble( parameterIndex: 15, p.getTrattamento().getPrezzo())
    attributi[14] = String.valueOf(p.getTrattamento().getPrezzo());
} else{
    preparedStatement.setNull( parameterIndex: 10, Types.VARCHAR);
    attributi[9] = "";
    preparedStatement.setNull( parameterIndex: 15, Types.VARCHAR);
    attributi[14] = "0";
}

preparedStatement.setString( parameterIndex: 11, p.getNoteAggiuntive().trim());
attributi[10] = p.getNoteAggiuntive().trim();
preparedStatement.setString( parameterIndex: 12, p.getTipoDocumento().trim());
attributi[11] = p.getTipoDocumento().trim();
preparedStatement.setBoolean( parameterIndex: 13, p.getStatoPrenotazione());
attributi[12] = String.valueOf(p.getStatoPrenotazione());
preparedStatement.setBoolean( parameterIndex: 14, p.isCheckIn());
attributi[13] = String.valueOf(p.isCheckIn());
preparedStatement.setString( parameterIndex: 16, p.getMetodoPagamento());
attributi[15] = String.valueOf(p.getMetodoPagamento());
preparedStatement.setString( parameterIndex: 17, p.getCittadinanza());
attributi[16] = String.valueOf(p.getCittadinanza());
preparedStatement.setInt( parameterIndex: 18,p.getIDPrenotazione());

```

9.4: Correzione proposta

TEST DI UNITÀ: WHITEBOX TC36

Titolo: Test di modifica della lista dei servizi di una prenotazione (aggiunta di un servizio). Si prova ad aggiungere un nuovo servizio richiesto da un cliente. In questo caso, questo è il primo servizio richiesto da un cliente; inoltre non ha richiesto alcun trattamento, ovvero questo è *null*.

Descrizione dell'errore:

L'errore lanciato è un *NullPointerException*, lanciato all'ultima istruzione quando di esegue *AssertNotThrow()*. Questa viene lanciata perché il campo trattamento di prenotazione è *null* e, nel metodo delegato ad effettuare il salvataggio delle modifiche alla prenotazione, viene invocato il metodo *getPrezzo()* sul trattamento (non verifica se questo sia stato modificato o meno).

Fix:

Basta tener conto della possibilità che il trattamento sia *null* ed impostare i campi della prenotazione in maniera adeguata (" " e 0 rispettivamente per i campi *nome* e *prezzo trattamento*).

```

@Test new *
@Tags({@Tag("Exception"), @Tag("Error")})
@DisplayName("TC36: doUpdate() quando la lista di servizi è vuota e Trattamento.getPrezzo() non è implementato")
public void doUpdateFalseListaServizi() throws SQLException{
    Prenotazione p = new Prenotazione();
    ArrayList<Cliente> clientes;
    ArrayList<Servizio> servizios;
    clientes = new ArrayList<>();
    clientes.add(new Cliente(nome: "Mario", cognome: "Rossi", provincia: "Roma", città: "Roma"));
    servizios = new ArrayList<>();
    Prenotazione prenotazione3 = new Prenotazione(LocalDate.of( year: 2023, month: 1, day: 1));
    prenotazione3.setMetodoPagamento("Contanti");
    assertDoesNotThrow(() -> prenotazioneDAO.doUpdate(prenotazione3));
}

```

10.1: Test che genera l'errore

```

org.opentest4j.AssertionFailedError: Unexpected exception thrown: java.lang.NullPointerException: Cannot invoke "it.unisa.Common.Trattamento.getPrezzo()"
<5 internal lines>
    at WhiteBox.UnitTest.prenotazioneDAOTesting.doUpdateFalseListaServizi(prenotazioneDAOTesting.java:238) <1 internal line>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1604)
<1 folded frame>
Caused by: java.lang.NullPointerException Create breakpoint : Cannot invoke "it.unisa.Common.Trattamento.getPrezzo()" because the return value of "it.unisa.Common.Trattamento.getPrezzo()" is null
    at it.unisa.Storage.DAO.PrenotazioneDAO.doUpdate(PrenotazioneDAO.java:476)
    at WhiteBox.UnitTest.prenotazioneDAOTesting.lambda$doUpdateFalseListaServizi$0(prenotazioneDAOTesting.java:238) <1 internal line>
... 6 more

```

10.2: Errore lanciato

```

attributi[3] = p.getDataInizio().toString();
preparedStatement.setDate( parameterIndex: 5, Date.valueOf(p.getDataFine()));
attributi[4] = p.getDataFine().toString();
preparedStatement.setDate( parameterIndex: 6, p.getDataEmissioneRicevuta() != null ?
attributi[5] = p.getDataEmissioneRicevuta() != null ? p.getDataEmissioneRicevuta()
preparedStatement.setString( parameterIndex: 7, p.getNumeroDocumento());
attributi[6] = p.getNumeroDocumento();
preparedStatement.setDate( parameterIndex: 8, Date.valueOf(p.getDataRilascio()));
attributi[7] = p.getDataRilascio().toString();
preparedStatement.setDate( parameterIndex: 9, Date.valueOf(p.getDataScadenza()));
attributi[8] = p.getDataScadenza().toString();

if(p.getTrattamento()!=null){
    preparedStatement.setString( parameterIndex: 10, p.getTrattamento().getNome().trim());
    attributi[9] = p.getTrattamento().getNome().trim();
}else{
    preparedStatement.setNull( parameterIndex: 10, Types.VARCHAR);

    attributi[9] = "";
}

preparedStatement.setString( parameterIndex: 11, p.getNoteAggiuntive().trim());
attributi[10] = p.getNoteAggiuntive().trim();
preparedStatement.setString( parameterIndex: 12, p.getTipoDocumento().trim());
attributi[11] = p.getTipoDocumento().trim();
preparedStatement.setBoolean( parameterIndex: 13, p.getStatoPrenotazione());
attributi[12] = String.valueOf(p.getStatoPrenotazione());
preparedStatement.setBoolean( parameterIndex: 14, p.isCheckIn());
attributi[13] = String.valueOf(p.isCheckIn());
preparedStatement.setDouble( parameterIndex: 15, p.getTrattamento().getPrezzo());
attributi[14] = String.valueOf(p.getTrattamento().getPrezzo());
preparedStatement.setString( parameterIndex: 16, p.getMetodoPagamento());
attributi[15] = String.valueOf(p.getMetodoPagamento());
preparedStatement.setString( parameterIndex: 17, p.getCittadinanza());
attributi[16] = String.valueOf(p.getCittadinanza());
preparedStatement.setInt( parameterIndex: 18,p.getIDPrenotazione());

```

10.3: Codice errato

```

attributi[6] = p.getNumeroDocumento();
preparedStatement.setDate( parameterIndex: 8, Date.valueOf(p.getDataRilascio()));
attributi[7] = p.getDataRilascio().toString();
preparedStatement.setDate( parameterIndex: 9, Date.valueOf(p.getDataScadenza()));
attributi[8] = p.getDataScadenza().toString();

if(p.getTrattamento()!=null){
    preparedStatement.setString( parameterIndex: 10, p.getTrattamento().getNome().trim());
    attributi[9] = p.getTrattamento().getNome().trim();
    preparedStatement.setDouble( parameterIndex: 15, p.getTrattamento().getPrezzo());
    attributi[14] = String.valueOf(p.getTrattamento().getPrezzo());
} else{
    preparedStatement.setNull( parameterIndex: 10, Types.VARCHAR);
    attributi[9] = "";
    preparedStatement.setNull( parameterIndex: 15, Types.VARCHAR);
    attributi[14] = "0";
}

preparedStatement.setString( parameterIndex: 11, p.getNoteAggiuntive().trim());
attributi[10] = p.getNoteAggiuntive().trim();
preparedStatement.setString( parameterIndex: 12, p.getTipoDocumento().trim());
attributi[11] = p.getTipoDocumento().trim();
preparedStatement.setBoolean( parameterIndex: 13, p.getStatoPrenotazione());
attributi[12] = String.valueOf(p.getStatoPrenotazione());
preparedStatement.setBoolean( parameterIndex: 14, p.isCheckIn());
attributi[13] = String.valueOf(p.isCheckIn());
preparedStatement.setString( parameterIndex: 16, p.getMetodoPagamento());
attributi[15] = String.valueOf(p.getMetodoPagamento());
preparedStatement.setString( parameterIndex: 17, p.getCittadinanza());
attributi[16] = String.valueOf(p.getCittadinanza());
preparedStatement.setInt( parameterIndex: 18, p.getIDPrenotazione());

```

10.4: Correzione proposta

Test Results prima...

Test Case	Test Result
TC1	FAIL
TC2	PASS
TC3	PASS
TC4	PASS
TC5	PASS
TC6	PASS
TC7	PASS
TC8	PASS

TC9	PASS
TC10	PASS
TC11	PASS
TC12	FAIL
TC13	PASS
TC14	PASS
TC15	PASS
TC16	PASS
TC17	FAIL
TC18	PASS
TC19	PASS
TC20	PASS
TC21	PASS
TC22	PASS
TC23	PASS
TC24	FAIL
TC25	FAIL
TC26	PASS
TC27	PASS
TC28	PASS
TC29	PASS
TC30	PASS
TC31	FAIL
TC32	FAIL
TC33	PASS
TC34	PASS
TC35	PASS

TC36	FAIL
TC37	PASS
TC38	PASS
TC39	PASS
TC40	PASS
TC41	PASS
TC42	PASS
TC43	PASS
TC44	PASS
TC45	PASS
TC46	PASS
TC47	PASS
TC48	PASS
TC49	PASS
TC50	PASS
TC51	PASS
TC52	PASS
TC53	PASS
TC54	PASS
TC55	PASS
TC56	PASS
TC57	PASS
TC58	PASS
TC59	PASS
TC60	PASS
TC61	PASS
TC62	PASS

TC63	PASS
------	------

... e dopo

Test Case	Test Result
TC1	PASS
TC2	PASS
TC3	PASS
TC4	PASS
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
TC25	PASS
TC26	PASS
TC27	PASS
TC28	PASS
TC29	PASS
TC30	PASS
TC31	PASS
TC32	PASS
TC33	PASS
TC34	PASS
TC35	PASS
TC36	PASS
TC37	PASS
TC38	PASS
TC39	PASS
TC40	PASS
TC41	PASS
TC42	PASS
TC43	PASS
TC44	PASS
TC45	PASS
TC46	PASS
TC47	PASS
TC48	PASS
TC49	PASS
TC50	PASS

TC51	PASS
TC52	PASS
TC53	PASS
TC54	PASS
TC55	PASS
TC56	PASS
TC57	PASS
TC58	PASS
TC59	PASS
TC60	PASS
TC61	PASS
TC62	PASS
TC63	PASS

TEST WHITEBOX – COVERAGE

Come specificato in precedenza, come tecnica di testing è stata applicata quella del branch testing di tipo whitebox. Di seguito sono riportati i risultati del path e branch coverage (rispettivamente 2° e 3° colonna):

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
ClienteDAO		98%		68%	15	31	4	162	1	9	0	1
PrenotazioneDAO		94%		73%	30	63	28	382	3	11	0	1
CameraDAO		89%		75%	12	33	13	132	1	9	0	1
DaoUtils		84%		61%	6	8	3	9	1	2	0	1
TrattamentoDAO		84%		60%	22	37	25	115	2	9	0	1
ServizioDAO		83%		61%	23	40	25	118	2	9	0	1

8.1: Risultati per i DAO

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Util		72%		81%	4	10	5	19	1	2	0	1
Ruolo		0%		n/a	1	1	4	4	1	1	1	1
Stato		100%		n/a	0	1	0	7	0	1	0	1

8.2: Risultati per il pacchetto Catalogue

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
FrontDesk		59%		75%	12	27	44	101	11	25	0	1

8.3: Risultati per il FrontDesk

TEST RESULT: INTEGRATION TESTS

Il passo successivo è il *test di integrazione*. Come strategia di testing è stato scelto un approccio di tipo Bottom-Up, scelta guidata principalmente dal tipo di architettura Client-Server implementata dal sistema; in particolare, si è scelto come livello più basso del testing (ovvero come punto “partenza”) quello relativo alla persistenza dei dati.

Nello specifico, sono stati individuati i tre seguenti livelli:

1. **logica di persistenza:** strato di persistenza (classi DAO) in congiunta con le classi che implementano i cataloghi (perché i cataloghi utilizzano i DAO direttamente);
2. **logica di business:** classi che implementano il pattern *command* e il pattern *observer* (entrambi fanno utilizzo dei cataloghi) integrate al livello 1;
3. **logica del client:** con chiamate al RMI registry connesso al RMI server + chiamate al DB, risultando in un vero e proprio test di sistema.

Si nota inoltre che l'aver effettuato sia testing *blackbox* che *whitebox* prima di quello d'integrazione si è rivelato molto profittevole, in quanto ha ridotto di molto gli errori riscontrati proprio durante la fase di test di integrazione, rendendo quest'ultima molto più agevole e veloce.

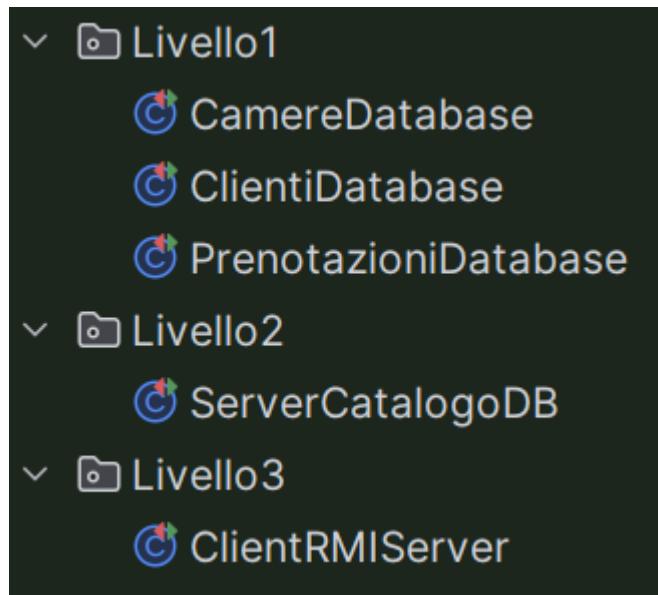


Figura 1: rappresentazione per livelli del test di integrazione

Test failures

TC4

Questo test simula la chiamata al database tramite il catalogo clienti (nello specifico viene simulato l'operazione di *retrieve* di un cliente dal Database).

Grazie all'esecuzione del test, è stato constatato che il catalogo dei clienti non recuperava l'operazione di retrieve correttamente, non recuperando alcuni dati.

```

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

```

8.1: Test nel quale si è riscontrato l'errore.

```

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
286   {

```

s ✘ 1 test failed, 3 passed 4 tests total, 2 sec 454 ms

s "C:\Program Files\Java\jdk-25\bin\java.exe" ...

s 17:00:48 DEBUG CatalogoClienti: DEBUG: Sto per chiamare doUpdate sul DB!

s 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 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){

```

8.2: correzione proposta

Test results prima...

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
TC 25	PASS
TC 26	PASS
TC 27	PASS
TC 28	PASS
TC 29	PASS
TC 30	PASS
TC 31	PASS
TC 32	PASS
TC 33	PASS
TC 34	PASS

... e dopo

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
TC 27	PASS
TC 28	PASS
TC 29	PASS
TC 30	PASS
TC 31	PASS
TC 32	PASS
TC 33	PASS
TC 34	PASS