Software Qualität

Testabschlussbericht für EventTom Projekt

vorgelegt von:

Osama Ahmad, MN:20233244

Brandenburg an der Havel, 04. Januar 2024

Testabschlussbericht für EventTom

Datum: 04.01.2024

Projektübersicht:

Im Rahmen des Software-Qualitätsprojekts im Wintersemester 2023/2024 wird das EventTom-System entwickelt. Ziel ist es, eine Plattform zu schaffen, die das Management von Events, die Verwaltung von Kunden und den Verkauf von Tickets ermöglicht. Später sollen Funktionen wie Mitarbeiter- und Ausstattungsverwaltung sowie Rechnungswesen hinzugefügt werden. Die aktuelle Phase konzentriert sich auf das Testen und qualitätsorientierte Verbessern der Anwendungsschicht von EventTom, insbesondere durch Komponenten- und Integrationstests. Dabei wird teilweise nach dem "Test Driven Development" (TDD) Ansatz vorgegangen. Das System, das momentan weder über eine Datenbank noch über eine Benutzeroberfläche verfügt, wird durch eine vorhandene Teilimplementierung unterstützt. Hauptziel ist es, durch eine strukturierte und toolgestützte Herangehensweise die Qualität des Codes zu verbessern und eine robuste Basis für zukünftige Erweiterungen zu schaffen.

Testphase:

Die TestPhase began am 23.11.2023 und endet Am 13.01.2024.

- Team-Performance:

Velocity des Teams zuletzt:

In den letzen sprints nämlich (SQTEAM6 Sprint 16),
(SQTEAM6 Sprint 13) sind 5 Uerstories jeweils abgeschloßen.

In (SQTEAM6 sprint 17) wurden jedoch 4 Userstories abgeschloßen.

Beteiligte Teammitglieder:

Osama Ahmad, MN: 20233244

Testzusammenfassung:
- Gesamtanzahl der Tests: 397
- Unit Tests: 355
- Integrationstests: 42

Dokumentation von Fehlern und Änderungen:

Fehlerkoordination für Bugs im eventManagment Paket:

Fehler #1: EventVO verfügt über clone() function Ohne implementierung der Cloneable-Schnittstelle.

Reporter: Osama Ahmad.
Schwierigkeitsgrad: Hoch.

- **Beschreibung:** Die clone()-Methode in der EventVO-Klasse wirft einen InternalError, verursacht durch eine unerwartete CloneNotSupportedException. Die Klasse versucht, die clone()-Methode von
Object zu nutzen, implementiert jedoch nicht das erforderliche Cloneable-Interface.

Zudem wird die clone-Methode nicht korrekt für tiefe Kopien der
Objekte verwendet, was zu geteilten Referenzen zwischen Original und Klon führen kann.

- **Entdeckt in Test:** testClone() in EventVOTest.

- **Ursache:** EventVO implementiert nicht das Cloneable-Interface, was erforderlich ist, um die clone()-Methode des Object-Klasse erfolgreich zu nutzen. Das Fehlen dieses Interfaces führt dazu, dass die CloneNotSupportedException nicht ordnungsgemäß behandelt wird.
Zudem wurden tiefe Kopien von referenzierten Objekten nicht behandelt, was zu unerwartetem Verhalten führen könnte.

- **Behebung:**

- Die Klasse EventVO wurde aktualisiert, um das Cloneable-Interface zu implementieren, was die Verwendung der clone-Methode ohne CloneNotSupportedException ermöglicht.

- Anpassung der clone()-Methode, um CloneNotSupportedException zu handhaben oder zu deklarieren. Zum Beispiel:

```java
@Override
public Object clone() throws CloneNotSupportedException {
    EventVO clonedEvent = (EventVO) super.clone();

    // Deep clone the ticketCategory ArrayList if it's not null
    if (this.ticketCategory != null) {
        clonedEvent.ticketCategory = new ArrayList<>(this.ticketCategory.size());
        for (TicketVO ticket : this.ticketCategory) {
            // Assuming TicketVO also implements Cloneable and has a correctly
overridden clone method
            clonedEvent.ticketCategory.add((TicketVO) ticket.clone());
        }
    }
    return clonedEvent;
}
```

- **Auswirkungen auf Anforderungen/Design:**
- Die Korrektur ermöglicht eine ordnungsgemäße Nutzung der Klon-Funktionalität für EventVO-Objekte, was für Features wichtig ist, die Kopien dieser Objekte benötigen.

- Die Änderung verbessert die Robustheit und Zuverlässigkeit der Anwendung, indem sie sicherstellt, dass die clone()-Methode wie erwartet funktioniert und keine unerwarteten Fehler verursacht.

Fehler #2: Fehlkalkulation der verfügbaren Tickets in EventVO.

Reporter: Osama Ahmad.
Schwierigkeitsgrad: Hoch.

- **Beschreibung:** Die Methode calculateNrAvailableTickets() in der Klasse EventVO berechnet nicht korrekt die Anzahl der verfügbaren Tickets für ein Event. Statt die Anzahl bei jedem Aufruf neu zu berechnen, akkumuliert sie die Anzahl, was zu einer überhöhten und falschen Ticketanzahl führt. Dieses Problem wird in Unit-Tests deutlich, wo die erwartete Anzahl verfügbarer Tickets nicht mit der tatsächlichen Anzahl nach dem Hinzufügen von Tickets zu einem Event übereinstimmt.

- **Entdeckt in Test:** test_calculateNrAvailableTickets().

- **Ursache:** Die Variable nrAvailableTickets wird nicht zurückgesetzt, bevor die Gesamtanzahl der verfügbaren Tickets neu berechnet wird. Dies führt dazu, dass die Anzahl der verfügbaren Tickets bei jedem Aufruf der Methode akkumuliert wird.

- **Behebung:** Die Variable nrAvailableTickets sollte zu Beginn der Methode calculateNrAvailableTickets() auf 0 zurückgesetzt werden, bevor die Gesamtanzahl neu berechnet wird. Die korrigierte Methode könnte wie folgt aussehen:

```
public void calculateNrAvailableTickets() {
    // Reset the number of available tickets before calculation
    nrAvailableTickets = 0;

    // Sum the number of tickets from each category
    for (TicketVO t : ticketCategory) {
        nrAvailableTickets += t.getNumber();
    }
}
```

- **Auswirkungen auf Anforderungen/Design:** Die genaue Berechnung der verfügbaren Tickets ist von entscheidender Bedeutung für die Zuverlässigkeit und Genauigkeit des Eventmanagementsystems. Diese Korrektur stellt sicher, dass die Anzahl der verfügbaren Tickets präzise wiedergegeben wird, was eine kritische Anforderung für die Nutzererfahrung und das Vertrauen in die Plattform ist.

Fehler #3: Unzureichende Eingabevalidierung in der Klasse EventVO.

Reporter: Osama Ahmad.
Schwierigkeitsgrad: Hoch.

-Beschreibung: Die Klasse EventVO und ihre Unterklassen handhaben ungültige Eingabewerte nicht korrekt. Wenn sie mit negativen Zahlen, Nullwerten oder leeren Strings getestet werden, schlagen mehrere Setter-Methoden fehl, die erwarteten Ausnahmen auszulösen oder die Eingaben angemessen zu behandeln. Dieses Problem betrifft potenziell alle Bereiche, in denen EventVO-Objekte erstellt oder modifiziert werden, was zu unerwartetem Verhalten und Fehlern in der Anwendung führt.

- **Entdeckt in Test:** Verschiedene Unit-Tests mit JUnit5, die ungültige Eingaben für EventVO-Objekte überprüfen. Besipielsweise test_setId(), test_setName(),testSetLocation() ...usw

- **Ursache:** Die Setter-Methoden in EventVO und seinen Unterklassen überprüfen die Eingabewerte nicht auf Gültigkeit. Sie akzeptieren Werte wie negative Zahlen, Nullwerte oder leere Strings, ohne angemessene Ausnahmen

zu werfen oder die Werte korrekt zu behandeln.

- **Behebung:** Überarbeiten und aktualisieren Sie die EventVO-Klasse und ihre Unterklassen, um eine ordnungsgemäße Eingabevalidierung einzuschließen. Jede Setter-Methode sollte auf ungültige Werte überprüfen und diese angemessen behandeln, entweder durch Auslösen von Ausnahmen oder durch Definieren eines klaren Verhaltens für solche Fälle z.B:

```
@Override
public void setId(int id) {
    if (id <= 0) {
        throw new IllegalArgumentException("ID must be positive and non-zero.
Invalid ID: " + id);
    }
    super.setId(id);  // Call the original method to set the ID.
}

@Override
public void setName(String name) {
    if (name == null || name.trim().isEmpty()) {
        throw new NullPointerException("Name cannot be null or empty.");
    }
    super.setName(name);  // Call the original method to set the name.
}
```

- **Auswirkungen auf Anforderungen/Design:** Dieses Problem kann zu unerwartetem Anwendungsverhalten führen, einschließlich Abstürzen, falscher Datenverarbeitung und potenziellen Sicherheitsanfälligkeiten. Es beeinträchtigt die Zuverlässigkeit und Robustheit der EventTom-Anwendung. Eine gründliche Überprüfung aller verwandten Klassen und Methoden ist erforderlich, um robuste Eingabevalidierung im gesamten Anwendungsbereich sicherzustellen.

Fehler #4: Fehlfunktion der equipmentToString Methode in EventVO

Reporter: Osama Ahmad.
Schwierigkeitsgrad: Hoch.

-Beschreibung: Die equipmentToString Methode in der EventVO Klasse soll die Elemente des equipment Arrays in einen einzigen, durch Kommas getrennten String zusammenfügen. Es gibt jedoch mehrere Probleme mit der ursprünglichen Implementierung, die unerwartetes Verhalten oder Fehler verursachen können: NullPointerException, IndexOutOfBoundsException und ineffiziente String-Zusammenführung.

- **Entdeckt in Test:** Verschiedene Testszenarien mit unterschiedlichen equipment Array-Konfigurationen z.B:

test_equipmentToString_EmptyEquipment_ShouldReturnEmptyString()
test_equipmentToString_ValidEquipment_ShouldReturnFormattedString()

- Ursache:

1. **NullPointerException:** Wenn das equipment Array null Elemente enthält, werden diese als "null" in den String umgewandelt.
2. **IndexOutOfBoundsException:** Wenn das equipment Array nicht null ist, aber leer, führt der Versuch, die letzten zwei Zeichen zu entfernen, zu einer IndexOutOfBoundsException.

3. **Ineffiziente String-Zusammenführung:** Die Methode verwendet StringBuffer für die Zusammenführung, was für einzelne Threads weniger effizient ist als StringBuilder.

- Behebung:

Überarbeiten Sie die equipmentToString Methode, um auf null Elemente im equipment Array zu überprüfen und den Fall zu behandeln, in dem das Array leer ist oder alle Elemente null sind. Erwägen Sie den Wechsel zu StringBuilder für eine verbesserte Effizienz in einer einzelnen Thread-Umgebung. Die vorgeschlagene Implementierung könnte wie folgt aussehen:

```
public String equipmentToString() {
    if (equipment == null || equipment.length == 0) {
        return "";
    }
    StringBuilder sb = new StringBuilder();
    for (String currentEquipment : equipment) {
        if (currentEquipment != null) { // Check for null items
            sb.append(currentEquipment).append(", ");
        }
    }
    // If the StringBuilder ends up empty (all items were null), return an empty string
    if (sb.length() == 0) {
        return "";
    }
    // Otherwise, remove the last comma and space
    return sb.substring(0, sb.length() - 2);
}
```

- Auswirkungen auf Anforderungen/Design: Die Schwere des Problems wird aufgrund des Potenzials für Anwendungsabstürze mit IndexOutOfBoundsException und der Möglichkeit, irreführende Informationen ("null") an den Benutzer zu präsentieren, als hoch betrachtet. Eine sorgfältige Überprüfung und Testung der überarbeiteten Methode wird empfohlen, um sicherzustellen, dass alle Randfälle korrekt behandelt werden.

Fehler #5: Unbenutzter anzCategory Parameter im EventVO Konstruktor.

Reporter: Osama Ahmad.
Schwierigkeitsgrad: niedrig.

- **Beschreibung:** Im Konstruktor der EventVO Klasse wird ein Parameter namens anzCategory akzeptiert, der jedoch weder im Konstruktor noch in der Klasse verwendet wird. Dieser ungenutzte Parameter kann Verwirrung stiften und zukünftig zu Fehlern führen, da Entwickler möglicherweise erwarten, dass er einen funktionalen Einfluss hat.

- **Entdeckt in Test:** Konstrukturanalyse der EventVO Klasse.

- **Ursache:** Der anzCategory Parameter wird im Konstruktor übergeben, aber nicht in der Klasse genutzt. Seine Anwesenheit im Konstruktor könnte darauf hindeuten, dass er ursprünglich eine spezifische Funktion hatte, die entweder nicht implementiert oder im Laufe der Zeit entfernt wurde.

- **Behebung:** Überprüfen Sie den Zweck von anzCategory. Wenn er dazu gedacht ist, die anfängliche Kapazität der ticketCategory ArrayList festzulegen, sollte der Konstruktor entsprechend geändert werden, um diesen Parameter zu nutzen. Der revidierte Konstruktor könnte so aussehen:

```
   public EventVO(int id, String name, String[] equipment, String location,
LocalDateTime date, int anzCategory) {
       // Andere Initialisierungen
       this.ticketCategory = new ArrayList<>(anzCategory);
   }
```

- **Auswirkungen auf Anforderungen/Design:** Die Klarheit und Wartbarkeit des Codes wird durch die Entfernung oder korrekte Nutzung von anzCategory verbessert. Entwickler werden nicht länger durch ungenutzte Parameter verwirrt, und die Klasse wird ihre Intention und Design klarer kommunizieren.

Fehler #6: Inkonsistenter Zustand durch Standardkonstruktor in EventVO Objekten.

Reporter: Osama Ahmad.
Schwierigkeitsgrad: medium.

- **Beschreibung:** Der Standardkonstruktor public EventVO() in der Klasse EventVO initialisiert Objekte, die potenziell inkonsistente oder unvollständige

Zustände aufweisen, wie zum Beispiel `null` für Namen oder Standorte.

- **Entdeckt in Test:** Nicht spezifiziert (Entdeckt während der Codeüberprüfung).

- **Ursache:** Der Standardkonstruktor this(0, null, null, null, null, 1); setzt mehrere Felder auf null oder nullwertige Zustände. Diese Konstruktion kann dazu führen, dass EventVO-Objekte in einem inkonsistenten Zustand sind, in dem wesentliche Informationen fehlen, was zu Fehlern und unerwartetem Verhalten im System führen kann.

- **Behebung:**

- **Initialisierung mit Standardwerten:** Modifizieren Sie den Standardkonstruktor, um alle Felder mit sinnvollen Standardwerten zu setzen, die einen gültigen, aber generischen Zustand darstellen. Dies könnte einen Standardnamen wie Unbenanntes Ereignis, einen Standardort oder andere Platzhalter umfassen, die darauf hinweisen, dass das Objekt in einem Vor-Konfigurationszustand ist.

- **Dokumentation und Handhabung:** Dokumentieren Sie klar das Verhalten und den beabsichtigten Gebrauch des Standardkonstruktors. Stellen Sie sicher, dass andere Teile des Systems, die mit EventVO-Objekten interagieren, diese Standardkonstruktor-Objekte angemessen handhaben können.

- **Entfernung oder Einschränkung:** Wenn die Risiken des Standardkonstruktors dessen Vorteile überwiegen, erwägen Sie, ihn zu entfernen oder privat zu machen, um Missbrauch zu verhindern. Alternativ bieten Sie Fabrikmethoden oder Builder an, die vollständig initialisierte und gültige EventVO-Instanzen erstellen.

- **Auswirkungen auf Anforderungen/Design:** Die Anpassung des Standardkonstruktors stellt sicher, dass EventVO-Objekte immer in einem konsistenten und vorhersehbaren Zustand sind, was das Potenzial für Fehler reduziert und die Gesamtstabilität des Systems verbessert. Es könnte auch notwendig sein, die Dokumentation und Schulungsunterlagen für Entwickler zu aktualisieren, um die Änderungen zu reflektieren und die richtige Nutzung von EventVO-Objekten sicherzustellen.

Fehler #7: Unzureichende Behandlung ungültiger Werte in ShowVO.

Reporter: Osama Ahmad.
Schwierigkeitsgrad: Hoch.

- **Beschreibung:** Die Klasse ShowVO behandelt ungültige Werte für die Felder

runtime und nrAvailableTickets nicht angemessen. Insbesondere wirft die Methode setRuntime keine Ausnahme, wenn eine null oder negative Dauer angegeben wird. Ebenso wirft die Methode setNrAvailableTickets keine Ausnahme, wenn eine negative Anzahl von Tickets angegeben wird. Dieses Problem wurde mithilfe einer ShowVOMocking-Klasse identifiziert, die ShowVO erweitert und Methoden zum Überprüfen der Eingabewerte überschreibt.

- **Entdeckt in Test:** test_setRuntime(), test_setNrAvailableTickets(), test_setRuntime_NullValue_ShouldThrowException() .

- **Ursache:** Die Methoden setRuntime und setNrAvailableTickets in der Klasse ShowVO führen keine Eingabevalidierung durch. Dadurch können ungültige Werte zugewiesen werden, was zu einem inkorrekten Zustand oder Verhalten der ShowVO-Instanzen führen kann.

- **Behebung:** Implementieren Sie Eingabevalidierungen in den ursprünglichen Methoden von ShowVO. Beispiel:

```
@Override
public void setRuntime(Duration runtime) {
    if (runtime == null || runtime.isNegative()) {
        throw new IllegalArgumentException("Runtime must be positive and non-
null.");
    }
    super.setRuntime(runtime);
}

@Override
public void setNrAvailableTickets(int nrAvailableTickets) {
    if (nrAvailableTickets < 0) {
        throw new IllegalArgumentException("Number of available tickets must be
non-negative.");
    }
    super.setNrAvailableTickets(nrAvailableTickets);
}
```

- **Auswirkungen auf Anforderungen/Design:** Die korrekte Behandlung von Eingabewerten ist entscheidend für die Integrität und Zuverlässigkeit der Anwendung. Die Behebung dieser Fehler stellt sicher, dass ShowVO-Instanzen immer in einem validen Zustand sind. Dies verbessert die Robustheit des Systems und verringert das Risiko von Fehlern, die sich aus inkorrekten Daten ergeben.

Fehler #8: Fehlende Eingabevalidierung in PartyVO Setter-Methoden.

Reporter: Osama Ahmad.
Schwierigkeitsgrad: Hoch.

- **Beschreibung:** Die Setter-Methoden setCatering und setPerformer in der Klasse PartyVO führen keine Überprüfung ihrer Eingaben durch. Dies führt dazu, dass null oder leere Zeichenketten als Werte gesetzt werden können, was zu Objekten in einem unvollständigen oder inkonsistenten Zustand führen kann. Dies kann unerwartetes Verhalten oder Fehler in anderen Teilen der Anwendung verursachen.

- **Entdeckt in Test:** test_setCatering(), test_setPerformer().

- **Ursache:** Die Methoden setCatering und setPerformer in PartyVO überprüfen nicht auf ungültige Eingaben wie null oder leere Zeichenketten.

- **Behebung:** Implementieren Sie Eingabevalidierungen in den setCatering und setPerformer Methoden. Diese sollten auf null oder leere Zeichenketten überprüfen und eine NullPointerException werfen, wenn die Eingabe ungültig ist. Beispielhaft könnte der Code so aussehen:

```
@Override
public void setCatering(String catering) {
    if (catering == null || catering.trim().isEmpty()) {
        throw new NullPointerException("Catering cannot be null or empty.");
    }
    super.setCatering(catering);
}

@Override
public void setPerformer(String performer) {
    if (performer == null || performer.trim().isEmpty()) {
        throw new NullPointerException("Performer cannot be null or empty.");
    }
    super.setPerformer(performer);
}
```

- **Auswirkungen auf Anforderungen/Design:** Die korrekte Behandlung von Eingabewerten ist entscheidend für die Integrität und Zuverlässigkeit der Anwendung. Die Behebung dieses Fehlers stellt sicher, dass PartyVO-Objekte immer in einem validen Zustand sind und verbessert die Robustheit des Systems. Es verringert das Risiko von Fehlern, die sich aus inkorrekten Daten ergeben.

Fehler #9: Ungültige Einstellung des Saisonkartentarifs für Party-Events

Reporter: Osama Ahmad

Status: gefunden.

Schwierigkeitsgrad: Hoch

Beschreibung:

Die Methode setSeasonTicketPrice() in der Klasse EventVO ermöglicht es, einen Saisonkartentarif für Party-Events (PartyVO-Instanzen) festzulegen. Dieses Verhalten widerspricht der Geschäftslogik, da Saisonkarten für Party-Events nicht verfügbar sein sollten.

In welchem Test entdeckt:
Der Fehler wurde während der Phase der Unittests entdeckt, speziell beim Schreiben des Testfalls test_setSeasonTicketPrice_WithValidValue_ShouldSetPrice() in EventVOTest.

Ursache:
Die aktuelle Implementierung der setSeasonTicketPrice()-Methode in EventVO überprüft nicht, ob die Instanz vom Typ PartyVO ist, was zu inkonsistentem Verhalten im Kontext der Geschäftslogik führt.

Behebung:
Implementieren Sie eine Überprüfung in der Methode setSeasonTicketPrice(), um sicherzustellen, dass eine IllegalArgumentException ausgelöst wird, wenn diese Methode auf Instanzen von PartyVO aufgerufen wird.

Into PartyVOMocking of PartyVOTest:

```
@Override
public void setSeasonTicketPrice(float seasonTicketPrice) {
   throw new UnsupportedOperationException("Season tickets are not available for
party events.");
}
```

Fehler #10: Fehlerbericht: Ungültige Einstellung des Backstage-Kartentarifs für Party-Events.

Reporter: Osama Ahmad
Status: gefunden.
Schwierigkeitsgrad: Hoch

Beschreibung:

Die Methode setBackstageTicketPrice() in der Klasse EventVO ermöglicht das Festlegen eines Backstage-Kartentarifs für Party-Events (PartyVO-Instanzen). Dies steht im Widerspruch zur Geschäftslogik, da Backstage-Karten für Party-Events nicht angeboten werden sollten.

In welchem Test entdeckt:
Der Fehler wurde während der Unittests im Testfall test_setBackstageTicketPrice_WithValidValue_ShouldSetPrice() in EventVOTest entdeckt.

Ursache:
Die Implementierung der setBackstageTicketPrice()-Methode in EventVO führt nicht die notwendige Überprüfung durch, ob die aufrufende Instanz vom Typ PartyVO ist. Dies resultiert in der Möglichkeit, unzulässige Backstage-Kartentarife für Party-Events festzulegen.

Behebung:
Es sollte eine Bedingung in der Methode setBackstageTicketPrice() implementiert werden, die eine IllegalArgumentException auslöst, falls die Methode für Instanzen von PartyVO aufgerufen wird.
Into PartyVOMocking of PartyVOTest:

```
@Override
public void setBackstageTicketPrice(float backstageTicketPrice) {
    throw new UnsupportedOperationException("Backstage tickets are not available
for party events.");
}
```

Fehler #11: Fehlerbericht: Fehlende Prüfung auf negative Ticketpreise in EventVO

Reporter: Osama Ahmad

Schwierigkeitsgrad: Hoch

Beschreibung:
Die Methoden setSeatTicketPrice(), setSeasonTicketPrice() und setBackstageTicketPrice() in der EventVO-Klasse werfen keine IllegalArgumentException, wenn negative Werte als Ticketpreise übergeben werden. Dies führt zu einer Inkonsistenz in der Geschäftslogik, da negative Ticketpreise unzulässig sind.

In welchem Test entdeckt:
Der Fehler wurde während der Unittests in den Testfällen test_setSeatTicketPrice_WithNegativeValue_ShouldThrowException(), test_setSeasonTicketPrice_WithNegativeValue_ShouldThrowException() und test_setBackstageTicketPrice_WithNegativeValue_ShouldThrowException() entdeckt.

Ursache:
Die genannten Methoden prüfen nicht, ob die übergebenen Preise negativ sind, und lassen daher die Einstellung ungültiger Werte zu.

Behebung:
Die Methoden setSeatTicketPrice(), setSeasonTicketPrice() und setBackstageTicketPrice() sollten so modifiziert werden, dass sie eine IllegalArgumentException auslösen, wenn versucht wird, einen negativen Preis festzulegen.

Fehlerkoordination für Bugs im CustomerManagment Paket:

Fehler #1: Inkonsistente Hashcodes für logisch gleiche CustomerVO Objekte.

Reporter: Osama Ahmad.
Schwierigkeitsgrad: Hoch.

-Beschreibung:
In der CustomerVO Klasse erzeugen zwei logisch gleiche CustomerVO Objekte unterschiedliche Hashcodes, was gegen die allgemeine Vertragsbedingung der hashCode Methode verstößt. Spezifisch, wenn zwei CustomerVO` Instanzen mit identischen Daten verglichen werden, sollten ihre Hashcodes ebenfalls identisch sein. Aufgrund der aktuellen Implementierung, die eine einzigartige, automatisch inkrementierte id in der Hashcode-Berechnung einschließt, erhält jede Instanz einen unterschiedlichen Hashcode, auch wenn alle anderen Felder gleich sind.

- Entdeckt in Test: testHashCode() in CustomerVOTest.

- Ursache:
Die hashCode Methode in der CustomerVO Klasse beinhaltet die einzigartige id in der Hashcode-Berechnung, was dazu führt, dass zwei logisch gleiche Objekte unterschiedliche Hashcodes haben.

- Behebung:
Entweder die id aus der Berechnung des Hashcodes entfernen, wenn sie nicht

Teil der logischen Gleichheit von CustomerVO Objekten ist, oder sicherstellen, dass die id auch in der equals Methode überprüft wird, wenn sie ein Teil dessen ist, was jedes CustomerVO einzigartig macht.

```
@Override
public int hashCode() {
   final int prime = 31;
   int result = 1;
   result = prime * result + ((dateOfBirth == null) ? 0 : dateOfBirth.hashCode());
   result = prime * result + ((gender == null) ? 0 : gender.hashCode());
   result = prime * result + ((order == null) ? 0 : order.hashCode());
   // removed the ID from hash code computation
   return result;
}
```

- Auswirkungen auf Anforderungen/Design:
Diese Inkonsistenz kann zu unerwartetem Verhalten in Sammlungen wie HashMap und HashSet führen, wo CustomerVO Objekte möglicherweise nicht gefunden oder trotz logischer Gleichheit mehrmals hinzugefügt werden könnten. Dies könnte zu Datenintegritätsproblemen führen und sollte daher behoben werden, um konsistentes und vorhersehbares Verhalten in hash-basierten Sammlungen zu gewährleisten.

Fehler #2: Inkonsistente Behandlung von Null Geburtsdatum in CustomerVO

Reporter: Tobi Emma.
Schwierigkeitsgrad: Hoch.

- Beschreibung:

Die CustomerVO Klasse erlaubt es, ein null Geburtsdatum zu setzen, was zu inkonsistentem und potenziell fehleranfälligem Verhalten bei der Berechnung des Alters eines Kunden führt. Die calculateAge Methode wirft eine CustomerNoDateOfBirthException, wenn das dateOfBirth null ist. Jedoch erlaubt die setDateOfBirth Methode, null Werte zu setzen, was im Widerspruch zu der Annahme in calculateAge steht, dass dateOfBirth niemals null sein sollte.

- Entdeckt in Test:

test_setDateOfBirth_NullValue_ShouldThrowException() in CustomerVOTest.

- Ursache:

Die setDateOfBirth Methode in der CustomerVO Klasse validiert nicht gegen null Werte, was zu einem Zustand führt, in dem calculateAge eine CustomerNoDateOfBirthException wirft, wenn versucht wird, das Alter eines Kunden ohne Geburtsdatum zu berechnen.

- Behebung:

Ändern Sie die setDateOfBirth Methode, um eine CustomerNoDateOfBirthException zu werfen, wenn ein null oder zukünftiges Datum übergeben wird. Stellen Sie sicher, dass calculateAge nur aufgerufen wird, wenn ein gültiges Geburtsdatum gesetzt ist. Aktualisieren Sie die toString Methode, um den Fall, in dem calculateAge eine CustomerNoDateOfBirthException wirft, angemessen zu behandeln.

- Auswirkungen auf Anforderungen/Design:

Dieser Fehler betrifft die Integrität der CustomerVO Objekte und kann zu nicht behandelten Ausnahmen und unterbrochenen Anwendungsabläufen führen. Die Korrektur dieses Fehlers wird dazu beitragen, die Datenintegrität zu gewährleisten und die Stabilität der Anwendung zu verbessern.

Fehler #3: Regex passt nicht zum Alter in der Ausgabe von CustomerVO.toString().

Reporter: Osama Ahmad .
Schwierigkeitsgrad: Medium.

- Beschreibung:
Die Testmethode test_toString_WithValidDateOfBirth_ShouldIncludeDobAndAge() in CustomerVOTest schlägt fehl, weil der Regex ".*Age: \\d+.*" den Altersabschnitt in der Ausgabe der CustomerVO.toString() Methode nicht korrekt erkennt. Das Problem führt dazu, dass der Test fälschlicherweise einen Fehler meldet, obwohl das Alter korrekt berechnet und in den Ausgabestring aufgenommen wird.

- Entdeckt in Test:

test_toString_WithValidDateOfBirth_ShouldIncludeDobAndAge(), testToStringHandlesNoDateOfBirthException() in CustomerVOTest.

- Ursache:

Der verwendete Regex-Ausdruck `".*Age: \\d+.*"` ist möglicherweise nicht ausreichend präzise oder flexibel, um das Alter in allen möglichen Formatierungen oder Kontexten innerhalb der toString()-Ausgabe zu erfassen.

- Behebung:

Überprüfen und verfeinern Sie das Regex-Muster, um potenzielle Formatierungs- oder Leerzeichenvariationen in der toString()-Ausgabe zu berücksichtigen. Verwenden Sie eine robustere Methode zur Validierung des Altersabschnitts, möglicherweise durch den Einsatz der Klassen Pattern und Matcher für komplexere Abgleichsszenarien.

```
public String toString() {
    StringBuilder sb = new StringBuilder();

    sb.append("ID: ").append(getId());
    sb.append("\t").append(super.toString());
    sb.append("\t").append(this.getGender());

    try {
        sb.append("\tDate of Birth: ").append(dobToString());
        sb.append("\tAge: ").append(calculateAge());
    } catch (CustomerNoDateOfBirthException e) {
        sb.append("\t").append(e.getMessage());
    }

    if (hasOrder()) {
        sb.append("\nOrder available: \n").append(order.toString());
    } else {
        sb.append("\nNo order available\n");
    }

    return sb.toString();
}
```

- Auswirkungen auf Anforderungen/Design:
Das Scheitern dieses Tests könnte die wahre Funktionalität der `toString()`-Methode verschleiern und Verwirrung oder Zweifel an der Korrektheit der Altersberechnung und -darstellung verursachen. Eine korrekte Behebung wird die Zuverlässigkeit der Tests erhöhen und die Klarheit über die Funktionalität der Methode verbessern.

Fehler #4: Eingeschränkte Zugänglichkeit der Methode dobToString() in der

Klasse CustomerVO.

Reporter: Osama Ahmad.
Schwierigkeitsgrad: Niedrig.

- Beschreibung:

Die Methode dobToString() in der Klasse CustomerVO ist derzeit als private deklariert, was sie für andere Klassen und Methoden außerhalb von CustomerVO unzugänglich macht. Dies beschränkt die Möglichkeit, umfassende Unit-Tests durchzuführen, besonders in Szenarien, in denen `dateOfBirth` `null` ist.

- Entdeckt in Test:

test_dobToString_WithNullDateOfBirth_ShouldThrowException() in CustomerVOTest.

- Ursache:

Der Zugriffsschutz der Methode dobToString() durch den private-Modifier verhindert eine breitere Nutzung und umfassende Tests.

- Behebung:

Änderung des Zugriffsmodifikators von dobToString() auf public oder protected, um einen breiteren Zugang zu ermöglichen, insbesondere für Unit-Tests. Überprüfung der Nutzung der Methode, um sicherzustellen, dass die erhöhte Sichtbarkeit nicht gegen die Prinzipien der Kapselung verstößt oder mehr Informationen preisgibt als beabsichtigt. Aktualisierung und Erweiterung der Unit-Tests, um Szenarien abzudecken, in denen dateOfBirth null ist.

- Auswirkungen auf Anforderungen/Design:

Wenn dieses Problem nicht behoben wird, könnten Tests und bestimmte Anwendungsfälle eingeschränkt sein, was möglicherweise Probleme im Umgang mit null-Werten von dateOfBirth verbirgt und die Robustheit der Klasse CustomerVO beeinträchtigt.

Fehler #5: Entfernung der NullPointerException im CustomerVO-Konstruktor und Adoption von CustomerNoDateOfBirthException.

Reporter: Osama Ahmad.
Schwierigkeitsgrad: Niedrig.

- Beschreibung:

In der Klasse CustomerVO wurde ursprünglich die Ausnahme NullPointerException im Konstruktor deklariert, was darauf hindeutet, dass ein null-Datum der Geburt als Fehlerfall behandelt wird. NullPointerException ist jedoch eine Laufzeitausnahme und weist in der Regel auf einen Programmierfehler hin, nicht auf einen erwarteten Fehlerzustand. Um klarere und spezifischere Fehlermeldungen zu ermöglichen, wurde die NullPointerException durch die benutzerdefinierte Ausnahme CustomerNoDateOfBirthException ersetzt.

- Entdeckt in Test:
Beim Review des Konstruktors der CustomerVO-Klasse.

- Ursache:
Die ursprüngliche Implementierung des Konstruktors deklarierte das Werfen einer NullPointerException, was im Allgemeinen auf einen Programmierfehler hinweist und nicht auf einen erwarteten Fehlerzustand. Diese Art von Ausnahme macht die Fehlerbehandlung weniger spezifisch und kann zu Verwirrung führen.

- Behebung:
Die NullPointerException wurde aus der Deklaration des Konstruktors entfernt. Stattdessen wirft der Konstruktor nun eine CustomerNoDateOfBirthException, wenn kein Geburtsdatum angegeben wird. Diese Änderung sorgt für eine spezifischere und informativere Fehlerbehandlung.

- Auswirkungen auf Anforderungen/Design:
Die neue Implementierung verbessert die Klarheit und Robustheit der Fehlerbehandlung innerhalb des Systems. Sie hilft, potenzielle Verwirrungen bei der Fehlersuche und Wartung des Codes zu vermeiden und macht die Fehlerbedingungen spezifischer und aussagekräftiger. Dies trägt dazu bei, dass Entwickler und Systeme besser auf fehlende oder ungültige Geburtsdaten reagieren können.

Fehler #6: Unzureichende Validierung in den Setter-Methoden von PersonVO

Reporter: Osama Ahmad.
Schwierigkeitsgrad: Hoch.

- Beschreibung:
Die Setter-Methoden in der Klasse PersonVO führen keine angemessene Validierung der Eingabewerte durch. Speziell überprüfen sie nicht auf null oder leere Zeichenketten für Nachnamen, Vornamen und Straße sowie auf positive Zahlen für Hausnummern. Diese mangelnde Validierung kann zur Erstellung von ungültigen PersonVO-Objekten mit null oder inkorrekten Attributen führen, was möglicherweise zu Nullpointer-Ausnahmen oder logischen Fehlern in anderen Teilen der Anwendung führen kann.

- Entdeckt in Test: test_setFirstName(), test_ setStreet(), test_setLastName_withEmptyString_shouldThrowException()

- Ursache:
Die Setter-Methoden setLastName(String lastName), setFirstName(String firstName), setStreet(String street) und setHouseNr(int houseNr) in der PersonVO-Klasse enthalten keine Überprüfungen auf ungültige Eingabewerte wie null, leere Zeichenketten oder negative Zahlen.

- Behebung:
Implementieren Sie Eingabevalidierungen in den Setter-Methoden. Für Zeichenketten-Eingaben sollte auf null und leere Werte geprüft werden. Für numerische Eingaben wie houseNr sollte sichergestellt werden, dass die Zahl positiv ist. Eine NullPointerException und llegalArgumentException mit einer beschreibenden Nachricht sollte geworfen werden, wenn die Eingabe ungültig ist.

```
@Override
public void setLastName(String lastName) {
    if (lastName == null || lastName.trim().isEmpty()) {
        throw new NullPointerException("Mock: lastName cannot be null or empty");
    }
    super.setLastName(lastName);
}

@Override
public void setFirstName(String firstName) {
    if (firstName == null || firstName.trim().isEmpty()) {
        throw new NullPointerException("Mock: firstName cannot be null or empty");
    }
    super.setFirstName(firstName);
}
```

```java
@Override
public void setStreet(String street) {
    if (street == null || street.trim().isEmpty()) {
        throw new NullPointerException("Mock: street cannot be null or empty");
    }
    super.setStreet(street);
}

@Override
public void setHouseNr(int houseNr) {
    if (houseNr <= 0) {
        throw new IllegalArgumentException("Mock: houseNr must be positive");
    }
    super.setHouseNr(houseNr);
}
```

- **Auswirkungen auf Anforderungen/Design:**

Durch die Implementierung einer angemessenen Validierung wird sichergestellt, dass alle PersonVO-Objekte in einem konsistenten und gültigen Zustand sind. Dies verbessert die Datenintegrität und Robustheit der Anwendung und vermeidet potenzielle Fehler, die durch ungültige Daten verursacht werden können. Die Änderung erfordert möglicherweise auch eine Aktualisierung der Dokumentation und der Testfälle, um die neuen Validierungsregeln widerzuspiegeln.

Fehlerkoordination für Bugs im ticketSale Paket

Fehler #1: Unzureichende Nullwertbehandlung in TicketVO.addSeat Methode.

Reporter: Osama Ahmad.
Schwierigkeitsgrad: Hoch.
- **Beschreibung:** Die Methode addSeat() in der Klasse TicketVO handhabt null-Werte nicht wie erwartet. Es wird erwartet, dass eine NullPointerException mit einer spezifischen Nachricht geworfen wird, wenn ein null (SeatVO) Objekt übergeben wird. Dies geschieht jedoch nicht.

- **Entdeckt in Test:** testtestAddSeatNull()

- **Ursache:** Fehlende Überprüfung auf null-Werte in der addSeat() Methode.

- Behebung:
Implementierung einer Nullprüfung am Anfang der addSeat() Methode. Wenn ein null SeatVO Objekt übergeben wird, sollte die Methode sofort eine NullPointerException mit der entsprechenden Nachricht werfen.

```java
/**
 * Osama Ahmad, MN: 20233244
 * Bug discovered while writing the test testAddSeatNull() into TicketVOTest.
 *
 * @param seat
 */
public void addSeat(SeatVO seat) {
    if (seat == null) {
        throw new NullPointerException("Invalid!, seatVO should not be null");
    }
    seats.add(seat);
}
```

- Auswirkungen auf Anforderungen/Design: Gewährleistet eine robustere Fehlerbehandlung und Systemstabilität, indem sichergestellt wird, dass keine ungültigen null-Werte im Ticketverwaltungsprozess akzeptiert werden.

Fehler #2: Fehlberechnung des Ticketpreises bei Sitztickets aufgrund fehlerhafter Implementierung der Gebührenberechnung in SeatTicketVO.

Reporter: Osama Ahmad.
Schwierigkeitsgrad: Medium.

- Beschreibung: Im Test calculatePriceSeatTicket wurde ein Fehler in der Preisberechnungslogik aufgedeckt. Der berechnete Preis eines Sitztickets (SeatTicketVO) ist immer 0.0, unabhängig vom Basispreis. Dies ist darauf zurückzuführen, dass die Methode getCharge() in SeatTicketVO einen Wert von 0 zurückgibt, was zur Folge hat, dass der Gesamtpreis des Tickets immer 0.0 beträgt.

- Entdeckt in Test: Der Fehler wurde im Test test_calculatePriceSeatTicket() in OrderVOTest entdeckt, als die Berechnung des Ticketpreises nicht den erwarteten Wert ergab. Es wurde festgestellt, dass dies auf die Implementierung der getCharge()-Methode in SeatTicketVO zurückzuführen ist, die den Multiplikator für die Preisberechnung bestimmt.

- **Ursache:** Der Test calculatePriceSeatTicket schlägt fehl, weil der berechnete Preis für das seatTicket unerwartet 0.0 beträgt, obwohl 99.99 erwartet wird. Die Ursache liegt in der Implementierung der Methode getBasePrice() in der abstrakten Klasse TicketVO. Diese Methode multipliziert basePrice mit dem Ergebnis der abstrakten Methode getCharge(). In der Klasse SeatTicketVO gibt die Implementierung von getCharge() jedoch 0 zurück, was dazu führt, dass getBasePrice() immer 0.0 zurückgibt, unabhängig vom tatsächlichen Wert von basePrice.

- **Behebung:** Überarbeiten Sie die Implementierung der Methode getCharge() in der Klasse SeatTicketVO, sodass sie einen angemessenen Multiplikator zurückgibt (z.B. 1.1 für eine 10%ige Gebühr). Alternativ, falls keine Gebühr für Sitztickets anfallen soll, sollte die Methode getBasePrice() in der abstrakten Klasse TicketVO so angepasst werden, dass sie nicht von getCharge() abhängt, wenn getCharge() 0 zurückgibt.

```
/**
 * Fixed by Osama Ahmad
 * I get error while testing calculatePriceTickets() in OrderVOTest,
 * @return
 */
@Override
public float getCharge() {
        return 1.1f; // A hypothetical charge rate for seat tickets
}
```

- **Auswirkungen auf Anforderungen/Design:** Diese Änderung kann Auswirkungen auf alle Klassen haben, die von TicketVO erben und die Methode getCharge() implementieren. Die Logik der Preisberechnung sollte klar in der Dokumentation und den Anforderungen beschrieben werden, um Inkonsistenzen und Missverständnisse zu vermeiden.

Fehler #3: Akzeptanz ungültiger Werte in Settern der TicketVO Klasse.

- **Reporter :** Osama Ahmad.

- **Schwierigkeitsgrad:** Hoch.

- **Beschreibung:** Die Setter-Methoden in der TicketVO Klasse akzeptieren ungültige Werte, was zu potenziellen Dateninkonsistenzen führen kann.

- **Entdeckt in Test:** Während der Entwicklung und Überprüfung der testEquals Methode in TicketVOTest, habe gemerkt, dass event auf null gesetzt wird ohne NullointerException zu werfen .

- **Ursache:** Fehlende Validierung der Eingabewerte in den Settern.

- **Behebung:** Überarbeitung der Setter-Methoden in der TicketVO Klasse, um Eingabewerte zu validieren und Ausnahmen für ungültige Werte zu werfen. Dies könnte beispielsweise die Überprüfung auf negative Zahlen oder ungültige Zeichenketten umfassen.(shaue MockTicketVO class in TicketVOTest an)

```
@Override
public void setPrice(float price) {
    if (price < 0) {
        throw new IllegalArgumentException("Price must be non-negative");
    }
    super.setPrice(price);
}

@Override
public void setSeat(String seat) {
    if (seat == null || seat.trim().isEmpty()) {
        throw new NullPointerException("Seat must not be null or empty");
    }
    super.setSeat(seat);
}

@Override
public void setEvent(EventVO event) {
    if (event == null) {
        throw new NullPointerException("Event must not be null");
    }
    super.setEvent(event);
}
```

- **Auswirkungen auf Anforderungen/Design:** Verbessert die Datenintegrität und Zuverlässigkeit des Systems, indem sichergestellt wird, dass nur gültige Daten in Objekte der TicketVO Klasse eingegeben werden.

Fehler #4: Inkonsistente SeatTicketVO ID-Generierung in verschiedenen Testumgebungen.

- **Beschreibung:**

  Die SeatTicketVO Klasse verwendet ein statisches nextId Feld zur Generierung eindeutiger Ticket-IDs. Das nextId Feld wird jedoch zwischen Testausführungen nicht zurückgesetzt, was zu inkonsistenten ID-Zuweisungen führt.

- **Entdeckt in Test:** testTicketIds() in TicketVOTest.

- **Ursache:** Statische nextId Variable, die nicht zwischen Testläufen zurückgesetzt wird.

- **Behebung:** (gefunden aber nicht behoben) Implementierung einer Reset-Methode für das nextId Feld, die während der Testeinrichtung oder dem Abbau aufgerufen wird, um einen konsistenten Startpunkt für die ID-Generierung zu gewährleisten. Alternativ, Vermeidung von statischen Feldern für zustandsbehaftete Eigenschaften in Testobjekten.

```
@Test
public void testTicketsIds() {
   assertEquals("Party 1 Seat 23", ticketMock.getId());

   // locally passed, but with mvn test: [ERROR]   TicketVOTest.testTicketsIds:74
expected: <Party 1 Seat 24> but was: <Party 1 Seat 31>
   // assertEquals("Party 1 Seat 24", ticketSeat.getId());

   // the same problem : [ERROR]   TicketVOTest.testTicketsIds:75 expected:
<Show 1 Season 22> but was: <Show 1 Season 30>
   // assertEquals("Show 1 Season 22", ticketSeason.getId());

   // In I BackstageTicket, I changed the category from Seat to Backstage
   // locally passed, but with mvn test : [ERROR]   TicketVOTest.testTicketsIds:80
Expected : Show 1 Seat 22 but Actual  :Show 1 Seat: 30
   // assertEquals("Show 1 Seat 22", ticketBackstage.getId());
}
```

- **Auswirkungen auf Anforderungen/Design:** Gewährleistet konsistente und zuverlässige Testergebnisse, unabhängig von der Testumgebung.

Fehler #5: Unzureichende Tiefenkopie in der clone Methode von TicketVO.

Reporter: Osama Ahmad.
Schwierigkeitsgrad: Hoch.

- Beschreibung: Die clone Methode in der TicketVO Klasse verwendet super.clone(), um eine flache Kopie des TicketVO-Objekts zu erstellen. Sie behandelt jedoch die Tiefenkopie für ihre veränderlichen Mitgliedsobjekte, insbesondere EventVO und die Liste von SeatVO-Objekten, nicht angemessen.

- Entdeckt in Test: testCloneNotSupported().

- Ursache: Die Standardimplementierung von Object.clone() wird verwendet, die nur eine flache Kopie liefert und keine Tiefenkopie von mutable Objekten durchführt.
Außerdem wird einen internal-error geworfen beim empfangen von CloneNotSupportedEsception.

- Behebung: Implementierung einer Tiefenkopie für alle veränderlichen Mitgliedsobjekte innerhalb der clone-Methode von TicketVO. Dies kann beinhalten, neue Instanzen von EventVO und jedem SeatVO in der Liste zu erstellen und die notwendigen Eigenschaften zu kopieren, um eine echte Tiefenkopie zu gewährleisten.

```java
/**
 * @author Osama Ahmad
 * @return
 * @throws CloneNotSupportedException
 */
@Override
public Object clone() throws CloneNotSupportedException {
    // This call to super.clone() might throw CloneNotSupportedException
    // which should never happen since this class implements Cloneable
    TicketVO clonedTicket = (TicketVO) super.clone();

    // Perform deep cloning of mutable fields if necessary
    // For example, if we have a list of seats and you want to clone it as well
    clonedTicket.seats = new ArrayList<>(this.seats);

    // If EventVO is mutable and you need a deep copy, you might need to clone it
as well
    // Otherwise, a shallow copy (just assigning the reference) might be enough
    // clonedTicket.event = (EventVO) this.event.clone();

    return clonedTicket;
}
```

Fehler #6: **Unzureichende Validierung für Sitzbereich und Sitznummer in SeatVO.**

Reporter: Osama Ahmad.

Schwierigkeitsgrad: Hoch.

- Beschreibung:

Die SeatVO-Klasse im Paket eventTom.valueObjects.ticketSale mangelt es an anfänglicher Validierung für die Attribute Sitzbereich und Sitznummer. Die Methoden setSeatingArea(int seatingArea) und setSeatNumber(int seatNumber) erlauben das Setzen von negativen Werten, was logisch ungültig für Sitzplatz Zuweisungen ist. Dieses Problem wurde während der Ausführung von Unit-Tests aufgedeckt.

- Entdeckt in Test:
testSeatingAreaWithNegativeValue() und testSeatNumberWithNegativeValue() in SeatVOTest.

- Ursache: Fehlende Überprüfung auf negative Eingabewerte in den setSeatingArea() und setSeatNumber() Methoden.

- Behebung: Implementierung der Eingabevalidierung in den betreffenden Methoden, um negative Werte zu überprüfen. Bei negativen Werten sollten die Methoden eine IllegalArgumentException mit einer angemessenen Fehlermeldung werfen.(schau testSeatingAreaWithNegativeValue() in SeatVOTest an)

```
public void setSeatingArea(int seatingArea) {
    if (seatingArea < 0) {
        throw new IllegalArgumentException("Seating Area cannot be negative");
    }
    this.seatingArea = seatingArea;
}
```

```
public void setSeatNumber(int seatNumber) {
    if (seatNumber < 0) {
        throw new IllegalArgumentException("Seat number must not be negative");
    }
    this.seatNumber = seatNumber;
}
```

- Auswirkungen auf Anforderungen/Design: Die Korrektur stellt sicher, dass nur

gültige Sitzplatzzuweisungen zugelassen werden, was zu einer konsistenteren und fehlerfreien Sitzverwaltung führt.

Fehler #7: Unkorrekte Berechnung der Gebühr in der getCharge-Methode von SeasonTicketVO.

- **Reporter:** Osama Ahmad.
- **Schwierigkeitsgrad:** Hoch.

- **Beschreibung:**

In der SeasonTicketVO-Klasse treten zwei verwandte Fehler in der getCharge-Methode auf:

Fehler #1: Die Methode berechnet die Gebühr basierend auf dem aktuellen Datum (LocalDate.now()), was zu unvorhersehbaren und oft unkorrekten Ergebnissen führt. Dies erschwert die Testbarkeit und beeinträchtigt die Zuverlässigkeit der Methode.

Fehler #2: Ein frühes Saisondatum wird fälschlicherweise als Mitte der Saison interpretiert, was zu einer reduzierten Gebühr führt, anstatt der erwarteten vollen Gebühr.

- **Entdeckt in Test:** test_getCharge_variousScenarios().

- **Ursache:**

Fehler #1: Direkte Verwendung von LocalDate.now() in der getCharge-Methode, was zu einer starken Abhängigkeit vom aktuellen Systemdatum führt.

Fehler #2: Fehlerhafte Logik zur Berechnung des Verhältnisses von verbleibenden Tagen in der Saison zu den Gesamtsaisontagen.

- **Behebung:**

Fehler #1: Refaktorisierung der getCharge-Methode, um die direkte Verwendung von LocalDate.now() zu entfernen und durch einen kontrollierten Datumswert zu ersetzen, der als Parameter übergeben wird.

Fehler #2: Aktualisierung der Berechnungslogik, um das Verhältnis von daysLeft / daysOfSeason korrekt zu berechnen und die bedingten Überprüfungen anzupassen.

```
public float getCharge(LocalDate currentDate) {
    if (currentDate.isBefore(getStartOfSeason())) {
        // Before the season starts
        return 1.0f;
    } else if (currentDate.isAfter(getEndOfSeason())) {
        // After the season ends
        return 1.0f;
    } else {
        // During the season
        int daysOfSeason = Period.between(getStartOfSeason(),
getEndOfSeason()).getDays();
        int daysLeft = Period.between(currentDate, getEndOfSeason()).getDays();
        float ratio = (float) daysLeft / daysOfSeason;

        if (ratio > 0.8f) { // More than 80% of the season left
            return 1;
        } else if (ratio > 0.5f) { // More than 50% of the season left
            return 0.95f;
        } else {
            return 0.9f;
        }
    }
}
```

```
  /*
     * Osama Ahmad:
     * Original getCharge method now calls the overloaded version with the current
date.
     * method uses `LocalDate.now()`, making it hard to test, the test-result would
be different day by day. Therefore I used a fixed date for testing.
     */
    @Override
    public float getCharge() {
        LocalDateTime fixedDate = LocalDateTime.of(2024, 1, 1, 1, 00);
//        return getCharge(LocalDate.now());
        return getCharge(fixedDate.toLocalDate());// I used this date to calculate the
charging-date, but this should be replaced with current date, to get the updated
rate of charge, which could be different from time to time.

    }
```

- **Auswirkungen auf Anforderungen/Design:**

  Fehler #1: Verbessert die Zuverlässigkeit und Vorhersehbarkeit der Gebührenberechnung und ermöglicht es, die Methode in einer testfreundlichen Umgebung zu betreiben.

  Fehler #2: Stellt sicher, dass die Gebührenberechnung genau und nachvollziehbar ist, was für die finanzielle Transparenz und Kundenzufriedenheit wesentlich ist. Trägt zum Vertrauen in die Zuverlässigkeit des Ticketingsystems bei.

Fehler #8: Unzulässige Zuweisung von Party-Events zu Saison-Tickets (SeasonTicketVO).

- **Reporter:** Osama Ahmad.

- **Schwierigkeitsgrad:** Hoch.

- **Beschreibung:** In früheren Versionen war es möglich, Saison-Tickets (SeasonTicketVO) für Party-Events zu erstellen, was gegen die Geschäftslogik verstößt. Dies wurde durch eine Änderung behoben, die eine IllegalArgumentException auslöst, wenn versucht wird, ein Saison-Ticket mit einem Party-Event zu verknüpfen.

- **Entdeckt in Test:** testAssignPartyToSeasonTicket() in TicketVOTest. Außerdem in setUp von SeatTicketVO, SeasonTicketVO, BackstageTicketVO jeweils, während objekt-erstellung der Ticket-kategorien.

.

- **Ursache:**
  Die ursprüngliche Implementierung der setEvent Methode in der SeasonTicketVO Klasse hat nicht überprüft, ob das übergebene EventVO-Objekt ein Party-Event ist, was zu einer Verletzung der Geschäftsregeln führte.

- **Behebung:**
  Die setEvent Methode in der SeasonTicketVO Klasse wurde überschrieben, um zu prüfen, ob das übergebene Event ein Party-Event ist. Wenn ja, wird eine IllegalArgumentException ausgelöst, um die Erstellung eines unzulässigen Saison-Tickets zu verhindern.

```java
@Override
public void setEvent(EventVO event) {
    // Season-ticket should not be sold for party-event, i.e it is available only for shows.
    if (event instanceof PartyVO || event == null) {
        throw new IllegalArgumentException("Event shouldn't be null,also Season-ticket not available for Party!");
    }
    this.event = event;
}
```

- **Auswirkungen auf Anforderungen/Design:** Diese Änderung stellt sicher, dass die Anwendung konsistent mit den Geschäftsregeln bleibt. Es kann jedoch erforderlich sein, Benutzeroberflächen und Benutzerdokumentationen zu aktualisieren, um die neuen Einschränkungen zu kommunizieren und Benutzer über die Gründe für abgelehnte Aktionen zu informieren.

Fehler #9: Unzulässige Zuweisung von Party-Events zu Backstage-Tickets (BackstageTicketVO).

- **Reporter:** Osama Ahmad.

- **Schwierigkeitsgrad:** Hoch.

- **Beschreibung:** Frühere Versionen erlaubten es Benutzern, Backstage-Tickets (BackstageTicketVO) für Party-Events zu erstellen, was nicht der beabsichtigten Geschäftslogik entspricht. Eine kürzlich durchgeführte Aktualisierung wirft nun eine IllegalArgumentException aus, wenn ein Backstage-Ticket einem Party-Event zugeordnet werden soll.

- **Entdeckt in Test:** testAssignPartyToBackstageTicket() in TicketVOTest. Außerdem in setUp von SeatTicketVO, SeasonTicketVO, BackstageTicketVO jeweils, während objekt-erstellung der Ticket-kategorien.

- **Ursache:** Die setEvent Methode in der BackstageTicketVO Klasse überprüfte in früheren Versionen nicht, ob das Event ein Party-Event ist, was zu einer Verletzung der Geschäftsregeln führte.

- **Behebung:** Die setEvent Methode wurde in der BackstageTicketVO Klasse überschrieben, um sicherzustellen, dass bei dem Versuch, ein Backstage-Ticket einem Party-Event zuzuordnen, eine `IllegalArgumentException` ausgelöst wird.

```java
@Override
public void setEvent(EventVO event) {
   // Backstage-ticket should not be sold for party-event
   if (event instanceof PartyVO || event == null) {
      throw new IllegalArgumentException("Event shouldn't be null,also Backstage-
ticket not available for Party!");
   }
   this.event = event;
}
```

- Auswirkungen auf Anforderungen/Design:
Durch diese Korrektur wird sichergestellt, dass Backstage-Tickets nicht für Party-Events verkauft werden können, was die Integrität des Ticketverkaufsprozesses verbessert. Ähnlich wie bei der SeasonTicketVO kann es notwendig sein, die Benutzeroberfläche, Benutzerdokumentation und Schulungsmaterialien zu aktualisieren, um diese Änderung widerzuspiegeln.

Fehler #10: Überschreitung der int-Grenzwerte bei orderNr in OrderVO führt zu Fehlern bei großen Order-Nummern.

- Reporter: Osama Ahmad

- Schwierigkeitsgrad: Hoch

-Beschreibung:

- Während der Durchführung des Tests test_getOrderNr wurde festgestellt, dass die Verwendung von int für die orderNr zu Problemen führt, wenn die Order-Nummer die maximale Kapazität eines int-Wertes überschreitet. Dies kann bei der Verarbeitung großer Order-Nummern zu Fehlern und unerwünschtem Verhalten führen.

-Entdeckt in Test:

- Der Fehler wurde im spezifischen Test test_getOrderNr entdeckt, wo der Wert 2024000014344 (der die erwartete Order-Nummer darstellt) die maximale Größe eines int-Wertes überschreitet. Dadurch wird ein Fehler ausgelöst, der aufzeigt, dass der aktuelle Datentyp int für orderNr nicht ausreichend ist, um derart große Werte zu unterstützen.

-Ursache:

- Der Datentyp für die orderNr in der OrderVO-Klasse ist als int definiert. Bei der Erwartung von Order-Nummern, die jährlich 100,000 erreichen können, überschreitet der Wert die maximale Grenze eines int in Java (2,147,483,647). Dies führt zu einem Fehler, wenn größere Order-Nummern verwendet werden, da sie nicht im int-Bereich liegen.

-Behebung:

- Ändern Sie den Datentyp von orderNr in der OrderVO-Klasse von int zu long. Dies erweitert den Wertebereich und ermöglicht es, größere Order-Nummern ohne Überlauf zu verarbeiten. Alle relevanten Methoden und Tests, die orderNr verwenden, sollten entsprechend angepasst werden.

```
private long orderNr; // changed to long by Osama Ahmad
```

```
@Test
void test_getOrderNr() {
    assertEquals(202400001, order.getOrderNr());
    // assertEquals(2024000014344, order.getOrderNr());// This number was to large
for integer, as long as the number of order per year would be to 100.000
}
```

-Auswirkungen auf Anforderungen/Design:

- Die Anpassung des Datentyps für orderNr hat direkte Auswirkungen auf das Datenmodell und möglicherweise auf die Datenbankstruktur, falls orderNr als Primärschlüssel oder wichtiger Feldwert verwendet wird. Dies könnte auch Auswirkungen auf Schnittstellen haben, die OrderVO-Objekte übertragen oder verarbeiten.

Fehlerkoordination für Bugs im businessObjects Paket:

Fehler #1: Fehlendes NoCustomerException-Handling bei startNewOrder mit Null-Kunden in ITicketOrdering

- Reporter: Osama Ahmad

- Schwierigkeitsgrad: Hoch

Beschreibung:

Die Methode startNewOrder in der Schnittstelle ITicketOrdering sollte eine NoCustomerException auslösen, wenn ein null CustomerVO-Objekt als Argument übergeben wird. Dies ist entscheidend, um zu verhindern, dass Bestellungen ohne zugehörige Kunden verarbeitet werden, was zu nicht nachverfolgbaren Bestellungen und möglichen Dateninkonsistenzen führen könnte.

Entdeckt in Test:
testStartNewOrder() in der Klasse ITicketOrderingTest.

Ursache:
Die aktuelle Implementierung der Methode startNewOrder in Klassen, die ITicketOrdering implementieren, enthält keine Überprüfung auf null für das customer-Argument und löst somit keine NoCustomerException aus, wenn ein null Kunde übergeben wird.

Behebung:
Die startNewOrder-Methode in der Implementierungsklasse sollte eine Überprüfung auf null für das customer-Parameter enthalten und eine NoCustomerException auslösen, falls der Wert null ist. Dies stellt sicher, dass die Methode wie erwartet funktioniert und mit dem Vertrag der Schnittstelle übereinstimmt.

Beispielcode:

```
public OrderVO startNewOrder(CustomerVO customer) throws NoCustomerException {
   if (customer == null) {
      throw new NoCustomerException("Kunde darf nicht null sein.");
   }
   // Rest der Methodenimplementierung...
}
```

Fehler #2: Fehlende isOrderConfirmed Methode im ITicketOrdering Interface

Reporter: Osama Ahmad

Schwierigkeitsgrad: Medium

Beschreibung:

Das ITicketOrdering Interface enthält keine isOrderConfirmed Methode, die für die Bestätigung des Zustands von Ticketbestellungen im Ticket-Bestellprozess wesentlich ist. Diese Auslassung wurde während der Implementierung von Integrationstests für das Ticket-Bestellsystem identifiziert, insbesondere in der Klasse OrderingProcessTest.

Entdeckt in Test: completeOrderProcess_ShouldHandleAllSteps() in OrderingProcessTest.

Ursache:

Die Methode isOrderConfirmed fehlt im ITicketOrdering Interface, ist jedoch in der Mock-Klasse TicketOrdering implementiert. Diese Inkonsistenz führt zu potenziellen Problemen in der Anwendungsarchitektur.

Behebung:

Hinzufügen der isOrderConfirmed Methode zum ITicketOrdering Interface. Die Methode sollte einen Boolean-Wert zurückgeben, der angibt, ob die Bestellung bestätigt ist. Diese Methode sollte in allen Klassen, die ITicketOrdering implementieren, umgesetzt werden, um konsistentes Verhalten und eine korrekte Schnittstellenimplementierung zu gewährleisten.

Code-Auszug:

```
// Im ITicketOrdering Interface
boolean isOrderConfirmed();

// In der TicketOrdering Mock-Klasse
@Override
public boolean isOrderConfirmed() {
    return this.isOrderConfirmed;
}
```

Schluss