Atividade 3: Exercitando TDD (Gerenciador de Vôos)

2. Considere que um mesmo passageiro não pode ser adicionado duas vezes ao mesmo voo.

- a. Crie casos de testes que exponham essa falha:

O código original (Fase 2) usa
passengers.add(passenger) (um ArrayList), que permite duplicatas. O teste abaixo falharia:

```
// Em AirportTest.java, dentro de EconomyFlightTest
@Nested
@DisplayName("When we have a usual passenger")
class UsualPassengerTest {
  private Passenger mike;
  @BeforeEach
  void setUp() {
    mike = new Passenger("Mike", false); // [cite: 675, 678]
  }

  @Test
  @DisplayName("Then you cannot add him to an economy flight more than once")
  public void testEconomyFlightUsualPassengerCannotBeAddedTwice() {
    // Setup
    economyFlight.addPassenger(mike);
    // Exercise (Tentar adicionar pela segunda vez) e Verify
    assertAll( () -> assertEquals(false, economyFlight.addPassenger(mike)),
      () -> assertEquals(1, economyFlight.getPassengers().size())
    );
  }
}

// (Testes similares seriam criados para BusinessFlight e PremiumFlight)
```

- b. Implemente a funcionalidade capaz de satisfazer esse caso de teste:
    - Nota: Esta implementação assume que a classe Passenger foi refatorada para implementar equals() e hashCode(), permitindo que contains() funcione corretamente.

Atividade 3: Exercitando TDD (Gerenciador de Vôos)

```
// Em EconomyFlight.java
@Override
public boolean addPassenger(Passenger passenger) {
  if (passengers.contains(passenger)) {
    return false; // Passageiro já existe
  }
  return passengers.add(passenger);
}

// Em BusinessFlight.java
@Override
public boolean addPassenger(Passenger passenger) {
  if (passenger.isVip()) {
    if (passengers.contains(passenger)) {
      return false; // Passageiro já existe
    }
    return passengers.add(passenger);
  }
  return false;
}

// (Lógica similar seria aplicada em PremiumFlight.java)
```

3. Considere que todo voo possua uma quantidade limite de passageiros, definida ao criar o voo. Crie casos de teste e implemente a funcionalidade.

- **a. Refatoração (RED - Alterar construtor e testes existentes):** Primeiro, alteramos a classe

Flight para aceitar a capacidade.

```
// Em Flight.java (classe abstrata)
public abstract class Flight {
  private String id;
  protected List<Passenger> passengers = new ArrayList<Passenger>();
  private int capacity; // Novo atributo

  public Flight(String id, int capacity) { // Construtor atualizado
    this.id = id;
```

```
    this.capacity = capacity;
  }

  public int getCapacity() {
    return capacity;
  }
  // ... resto da classe
}

// Em EconomyFlight, BusinessFlight, PremiumFlight (construtores)
public EconomyFlight(String id, int capacity) {
  super(id, capacity);
}

// Em AirportTest.java (Setup)
// ...
economyFlight = new EconomyFlight("1", 50); // Definindo um limite
businessFlight = new BusinessFlight("2", 20); // Definindo um limite
// ...
```

- b. Criar caso de teste (RED):

Criar um teste que tente exceder a capacidade.

```
// Em AirportTest.java, dentro de EconomyFlightTest
@Test
@DisplayName("Then you cannot add passengers over the flight capacity")
public void testEconomyFlightCannotExceedCapacity() {
  // Setup: Voo com capacidade 1
  Flight economyFlightLimit1 = new EconomyFlight("3", 1);
  Passenger p1 = new Passenger("P1", false);
  Passenger p2 = new Passenger("P2", false);

  // Exercise
  economyFlightLimit1.addPassenger(p1);
  // Verify (Tentar adicionar o segundo)
  assertAll(
    () -> assertEquals(false, economyFlightLimit1.addPassenger(p2)), // Espera 'false'
```

Atividade 3: Exercitando TDD (Gerenciador de Vôos)

```
    () -> assertEquals(1, economyFlightLimit1.getPassengers().size()) // Tamanho deve ser
1
  );
}
```

- c. Implemente a funcionalidade (GREEN/REFACTOR):

Atualizar os métodos addPassenger para verificar o limite.

Java

```
// Em EconomyFlight.java (combinando com a lógica de duplicata)
@Override
public boolean addPassenger(Passenger passenger) {
  if (passengers.size() >= getCapacity()) {
    return false; // Atingiu a capacidade
  }
  if (passengers.contains(passenger)) {
    return false; // Passageiro já existe
  }
  return passengers.add(passenger);
}

// Em BusinessFlight.java
@Override
public boolean addPassenger(Passenger passenger) {
  if (passengers.size() >= getCapacity()) {
    return false; // Atingiu a capacidade
  }
  if (passenger.isVip()) {
    if (passengers.contains(passenger)) {
      return false; // Passageiro já existe
    }
    return passengers.add(passenger);
  }
  return false;
}
```