# Contents

# Program UML

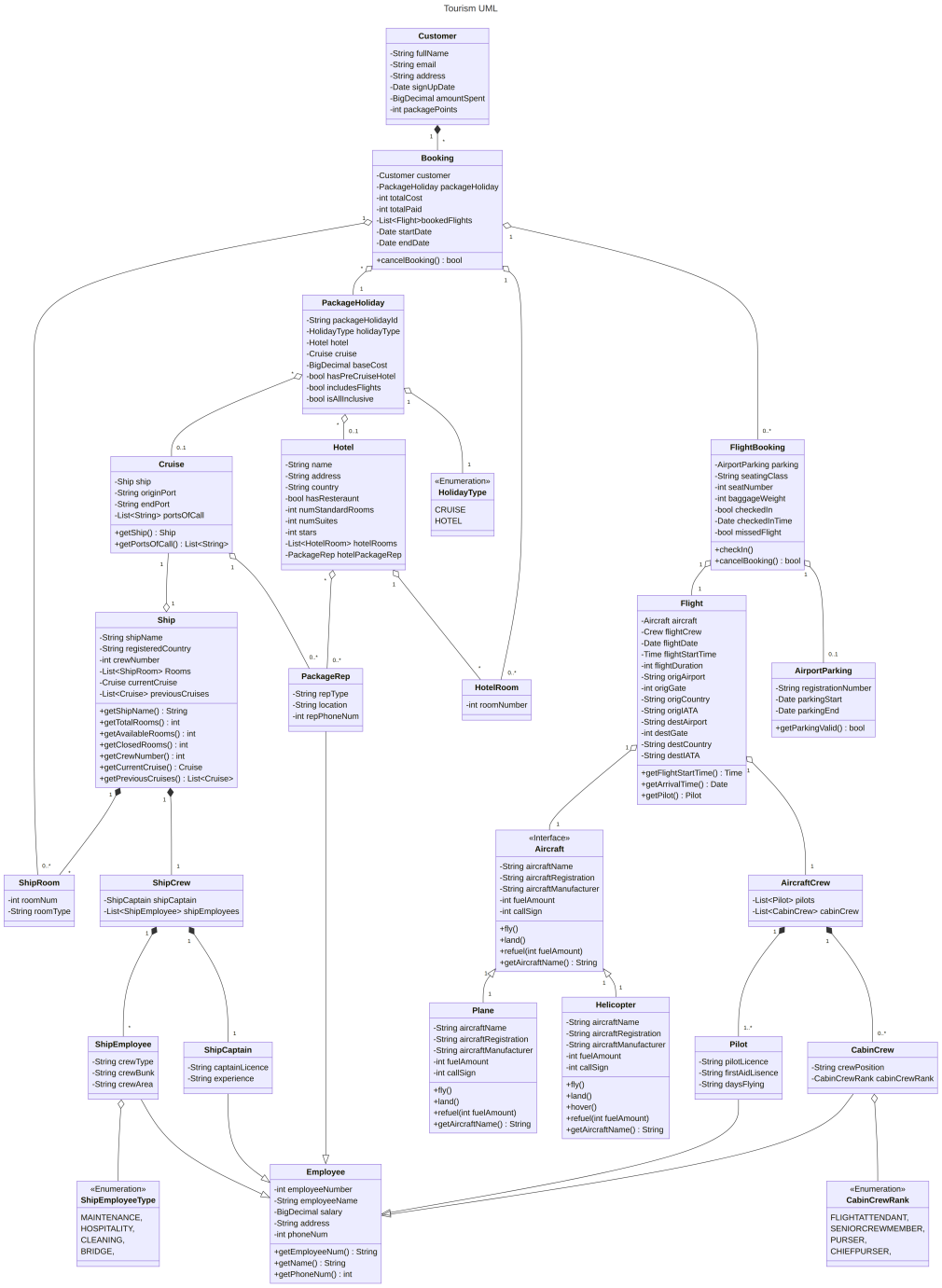


**Figure 1:** UML Representation of Program

First areas of testing.

The first stages of testing will ocurr during the development and programming process. The project will make extensive use of unit tests; these tests involve writing isolated automated tests that target small sections of the program, known as units.

Creation of the unit test involves developing a criteria, refered to the test case.

The unit returns output that is compared to the test's criteria that is known to be correct by the developers.

Ensuring that these small units of the program correct

Testing lots of small units that integrate to form a complex program reduces the amount of uncertain variables compared to a large monolithic test of the end result.

Furthermore,

Within this project, the Java unit testing framework JUnit will be used.

Building on top of unit testing, continuous integration ensures that all developers within the organisation:

- feature flags
- ui testing
- test lab
- testing pyramid
- e2e testing - operational - max interaction - require running services - automated ui testing server
- integration testing - testing api code inteactions, database connection
- black box testing - less interaction -
- martin folwer testing articles

Should the service require any external API's. These could be tracking APIs offered by various airports, or apis offered by the FIA (British equiavalent) to ensure that planes are correctly en route. Then the implementation of contact testing can ensure that updates to API returns and intefaces are quickly recognised and corrected. To ensure that the service does not suffer for too long.

End to end user testing.

`JUnit5` test for checking in process in `FlightBooking.java`,

```java
public boolean checkIn() {
    if (LocalDateTime.now().isAfter(flight.getFlightStart())) {
        hasMissedFlight = true;
        return false;
    }
    hasCheckedIn = true;
```

```
7       checkInTime = LocalDateTime.now();
8       return true;
9   }
```

The unit test created to test the `checkIn` method in the `FlightBooking` class.

```
1   package usw.pop;
2
3   import org.junit.jupiter.api.DisplayName;
4   import org.junit.jupiter.api.Test;
5   import org.mockito.Mockito;
6
7   import java.time.LocalDateTime;
8
9   import static org.junit.jupiter.api.Assertions.assertFalse;
10  import static org.junit.jupiter.api.Assertions.assertTrue;
11
12  class FlightBookingTest {
13      private Flight testFlight;
14      private FlightBooking testFlightBooking;
15
16
17      @Test
18      @DisplayName("Check-in process with an on-time flight booking")
19      public void checkIn() {
20          LocalDateTime earlyDateTime = LocalDateTime.now().plusYears(1);
21          /*
22           Create a mocked class of testFlight testFlight could use
23           an API for plane tracking, so creating a stub ensures that
24           the test is independent
25          */
26          testFlight = Mockito.mock(Flight.class);
27          // Set what to return when getFlightStart() is called
28          Mockito.when(testFlight.getFlightStart()).thenReturn(
29              earlyDateTime
30          );
31
32          // Call the check-in process
33          testFlightBooking = new FlightBooking(
34              testFlight, "economy", 12, 200
35          );
36          assertTrue(testFlightBooking.checkIn());
37      }
38
39      @Test
40      @DisplayName("Check-in process with a late flight booking")
41      public void checkInLate() {
42          LocalDateTime lateDateTime = LocalDateTime.now().minusYears(1);
43
44          // Create another mocked class
45          testFlight = Mockito.mock(Flight.class);
```

```
        // Set return time to the future to fail check-in
        Mockito.when(testFlight.getFlightStart()).thenReturn(
            lateDateTime
        );

        // Call the check-in process
        testFlightBooking = new FlightBooking(
            testFlight, "economy", 12, 200
        );
        assertFalse(testFlightBooking.checkIn());
    }
}
```

Narrow integration test with a PostgreSQL database.

```
package usw.pop;


import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.testcontainers.containers.PostgreSQLContainer;


/**
 * Test the program's integration with a database
 */
public class CustomerIntegrationTest {

    /*
     Create a temporary database container to test a database's
     integration with the program
    */
    static PostgreSQLContainer<?> postgres =
        new PostgreSQLContainer<>("postgres:16-alpine");

    Customer customer;
    // CustomerDatabase customerDatabase;

    /**
     * Start the database
     */
    @BeforeAll
    public static void startContainer() {
        postgres.start();
    }

    /**
     * Stop the database container
```

```
37          */
38         @AfterAll
39         public static void stopContainer() {
40             postgres.stop();
41         }
42
43         /**
44          * Connect to the database and proceed to clean it of previous
45          * entries
46          */
47         @BeforeEach
48         public void setUpCleanDb() {
49             customerDatabase = new CustomerDatabase(postgres.getJbdcUrl());
50             customerDatabase.deleteAll();
51         }
52
53         /**
54          * Test that {@code customerDatabase} integrates with and adds
55          * the customer to the database
56          */
57         @Test
58         @DisplayName("Database create customer")
59         public void dbCreateCustomer() {
60             customer = new Customer("Jake Real", "jakeemail@email.com",
61                 "3 Cool Road, Swandiff"
62             );
63             // Add the created customer to the database
64             customerDatabase.addCustomer(customer)
65             // Test that customer was created in the database
66             assertEquals("Jake Real", customerDatabase.getCustomerByName(
67                 "Jake Real").getFullName()
68             );
69         }
70
71         /**
72          * Test that {@code customerDatabase} integrates with and deletes
73          * the customer from the database
74          */
75         @Test
76         @DisplayName("Database delete customer")
77         public void dbDeleteCustomer() {
78             customer = new Customer("Jake Real", "jakeemail@email.com",
79                 "3 Cool Road, Swandiff"
80             );
81             // Add the created customer to the database
82             customerDatabase.addCustomer(customer);
83             // Test that customer was created in the database
84             assertEquals("Jake Real", customerDatabase.getCustomerByName(
85                 "Jake Real").getFullName()
86             );
87             // Delete the customer from the database
```

```
88          customerDatabase.deleteCustomerById(customer.getId);
89          // Check that there are no customers
90          assertEquals(0, customerDatabase.getAllCustomers().size());
91      }
92
93  }
```