 1. Higher Level Diagram:

Create a block diagram that shows the interaction between different sub-blocks of the system. In this case, you would likely have blocks representing `TravelPackage`, `Destination`, `Activity`, `Passenger`, and any other relevant components. Show how they interact with each other.

 2. Lower Level Diagram:

Create a UML class diagram that outlines the structure of the classes in your system. Include attributes and methods for each class. Identify relationships between classes, such as associations, aggregations, and compositions. Make sure to capture the hierarchy and dependencies.

Here is a simplified example:

plaintext
```
+-------------------+         +-------------------+        +----------------
---+
| TravelPackage    |          | Destination      |         | Activity
|
+-------------------+         +-------------------+        +----------------
---+
| -name: String    |1       * | -name: String    |1       * | -name: String
|
| -capacity: int   |---------| -activities: List|--------| -cost: double
|
| -itinerary: List |          |   of Activity    |         | -capacity: int
|
| -passengers: List|          +-------------------+        | -destination:
Destination|
+-------------------+                                      +----------------
---+

+-------------------+
| Passenger        |
+-------------------+
| -name: String    |
| -number: int     |
| -type: PassengerType |
| -balance: double |
| -activities: List|
+-------------------+
```

3. Implementation:

Implement the classes based on the UML diagram. Follow best practices for Java coding standards and encapsulation. Use appropriate data structures, and make sure to handle different passenger types accordingly.

```
public class TravelPackage {
    // Implementation here
}

public class Destination {
    // Implementation here
}

public class Activity {
    // Implementation here
}

public class Passenger {
    // Implementation here
}

public enum PassengerType {
    STANDARD, GOLD, PREMIUM
}
```

4. Unit Test Cases:

Write JUnit test cases for each class to ensure that they work as expected. Test different scenarios, including adding passengers, signing up for activities, checking balances, and ensuring that capacities are properly managed.

```
import org.junit.Test;
import static org.junit.Assert.*;

public class TravelPackageTest {
    // Unit tests for TravelPackage class
}

public class DestinationTest {
    // Unit tests for Destination class
}

public class ActivityTest {
    // Unit tests for Activity class
}

public class PassengerTest {
    // Unit tests for Passenger class
}
```