I can't directly draw the class diagram for you, but I can describe it in detail following the prompt and UML standards:

**Classes:**

* **Employee**
  + Attributes:
    - ID (int)
    - name (String)
    - dob (Date)
    - salary (double)
* **Programmer** (inherits from Employee)
* **AdministrativeEmployee** (inherits from Employee)
* **Project**
  + Attributes:
    - projectNumber (int) (unique identifier)
    - projectName (String)
    - status (String) (e.g., "In Progress", "Completed")
* **Report**
  + Attributes:
    - reportNumber (int) (unique identifier)
    - title (String)
    - reporterName (String)

**Relationships:**

* **Employee**
  + **Association** with **Project** (many-to-many): This association indicates that programmers can contribute to multiple projects, and a project can have multiple programmers working on it. You can represent this with a line connecting both classes and label it with "contributesTo" (verb phrase) at one end and "hasProgrammers" (verb phrase) at the other. **Consider using an additional class "ProjectAssignment"** (with attributes like projectID, employeeID) to represent this many-to-many relationship if your system requires keeping track of specific assignments.
* **Employee**
  + **Association** with **Report** (one-to-many): This association indicates that an administrative employee can create multiple reports, but a report can only have one creator. You can represent this with a line connecting Employee to Report with a crow's foot on the Report side labeled "createdBy".
* **Inheritance:**
  + **Programmer** inherits from **Employee** - This indicates that programmers are a specific type of employee and inherit all attributes and methods from the Employee class.
  + **AdministrativeEmployee** inherits from **Employee** - Similar to Programmer, this indicates that administrative employees are a specific type of employee.

**Additional Notes:**

* You can add methods (functions) to each class to represent their functionalities, but they were not specified in the prompt.

**Benefits of using a Class Diagram:**

* Provides a visual representation of the system's structure.
* Improves communication between developers and stakeholders.
* Helps identify potential design flaws early in the development process.

By following these guidelines, you can create a UML class diagram that accurately represents the structure of the ABC company's employee and project management system.

**مخطط الفئات لشركة ABC**

**الفئات:**

* **موظف**
  + **الصفات:**
    - رقم التعريف (int)
    - الاسم (String)
    - تاريخ الميلاد (Date)
    - الراتب (double)
* **مبرمج** (يرث من موظف)
* **موظف إداري** (يرث من موظف)
* **مشروع**
  + **الصفات:**
    - رقم المشروع (int) (معرف فريد)
    - اسم المشروع (String)
    - الحالة (String) (على سبيل المثال ، "قيد التقدم" ، "مكتمل")
* **تقرير**
  + **الصفات:**
    - رقم التقرير (int) (معرف فريد)
    - العنوان (String)
    - اسم المُقدم (String)

**العلاقات:**

* **موظف**
  + **ارتباط** مع **مشروع** (من متعدد إلى متعدد): يشير هذا الارتباط إلى أنه يمكن للمبرمجين المساهمة في مشاريع متعددة ، ويمكن أن يكون للمشروع العديد من المبرمجين الذين يعملون عليه. يمكنك تمثيل ذلك بخط يربط بين كلا الفئتين ووضع علامة عليه بـ "يساهم في" (عبارة الفعل) في أحد طرفيه و "لديه مبرمجون" (عبارة الفعل) في الطرف الآخر.
  + **ارتباط** مع **تقرير** (من واحد إلى متعدد): يشير هذا الارتباط إلى أنه يمكن للموظف الإداري إنشاء تقارير متعددة ، ولكن يمكن أن يكون للتقرير مُنشئ واحد فقط. يمكنك تمثيل ذلك بخط يربط بين الموظف والتقرير مع وجود قدم الغراب على جانب التقرير المُسمى "تم إنشاؤه بواسطة".
* **الإرث:**
  + **المبرمج** يرث من **الموظف** - يشير هذا إلى أن المبرمجين هم نوع محدد من الموظفين ويرثون جميع الصفات والطرق من فئة الموظف.
  + **الموظف الإداري** يرث من **الموظف** - على غرار المبرمج ، يشير هذا إلى أن الموظفين الإداريين هم نوع محدد من الموظفين.

**ملاحظات إضافية:**

* يمكنك إضافة طرق (وظائف) إلى كل فئة لتمثيل وظائفها ، ولكن لم يتم تحديدها في المطالبة.

**فوائد استخدام مخطط الفئات:**

* يوفر تمثيلًا بصريًا لهيكل النظام.
* يحسن التواصل بين المطورين وأصحاب المصلحة.
* يساعد على تحديد عيوب التصميم المحتملة في وقت مبكر من عملية التطوير.

باتباع هذه الإرشادات ، يمكنك إنشاء مخطط فئة UML يمثل بدقة بنية نظام إدارة الموظفين والمشاريع لشركة ABC.

تحتاج شركة تكنولوجيا تدعى شركة ABC إلى رسم رسم تخطيطي للفئات الخاص بالشركة. تمتلك الشركة العديد من الموظفين. يحتوي كل موظف على معرّف، اسم، تاريخ ميلاد وراتب. هناك نوعان من الموظفين: المبرمجين والموظفين الإداريين.

كل مبرمج يساهم في مشروع واحد على الأقل. يحتوي كل مشروع على رقم المشروع، اسم المشروع والحالة. كل موظف إداري يقوم بإنشاء تقرير واحد على الأقل. يحتوي كل تقرير على رقم التقرير، العنوان واسم الصحفي وهو اسم الموظف الذي قام بإعداد التقرير.

Employee class

n this expanded version, the Employee class includes the following attributes:

* id: Represents the ID of the employee. It is of type string.
* name: Represents the name of the employee. It is of type string.
* dob: Represents the date of birth of the employee. It is of type date.
* salary: Represents the salary of the employee. It is of type decimal.

The class also includes the following methods:

* getId(): Returns the ID of the employee.
* getName(): Returns the name of the employee.
* getDob(): Returns the date of birth of the employee.
* getSalary(): Returns the salary of the employee.
* setId(id: string): Sets the ID of the employee.
* setName(name: string): Sets the name of the employee.
* setDob(dob: date): Sets the date of birth of the employee.
* setSalary(salary: decimal): Sets the salary of the employee.

These methods allow for accessing and modifying the attributes of an employee object. By providing getter and setter methods, you can retrieve and update the employee's ID, name, date of birth, and salary as needed.

Programmer class :

In this expanded version, the Programmer class inherits from the Employee class and includes an additional attribute and methods specific to programmers:

Attributes:

* projects: Represents the list of projects that the programmer contributes to. It is of type List<Project>, where Project is another class that contains project-related information.

Methods:

* getProjects(): Returns the list of projects that the programmer contributes to.
* addProject(project: Project): Adds a project to the list of projects that the programmer contributes to.
* removeProject(project: Project): Removes a project from the list of projects that the programmer contributes to.

These additions allow for managing the projects associated with a programmer. The Programmer class inherits the attributes and methods from the Employee class, such as getId(), getName(), getDob(), getSalary(), setId(id: string), setName(name: string), setDob(dob: date), and setSalary(salary: decimal), to access and modify the basic employee information.

Report class:

In this expanded version, the Report class includes the following attributes:

* reportNumber: Represents the report number. It is of type string.
* title: Represents the title of the report. It is of type string.
* reporterName: Represents the name of the employee who created the report. It is of type string.

The class also includes the following methods:

* getReportNumber(): Returns the report number.
* getTitle(): Returns the title of the report.
* getReporterName(): Returns the name of the employee who created the report.
* setReportNumber(number: string): Sets the report number.
* setTitle(title: string): Sets the title of the report.
* setReporterName(name: string): Sets the name of the employee who created the report.

These methods allow for accessing and modifying the attributes of a Report object. By providing getter and setter methods, you can retrieve and update the report number, title, and reporter name as needed.

AdministrativeEmployee class:

1. Reports: The AdministrativeEmployee class has a private attribute called reports, which is a list of Report objects. This attribute represents the reports created by the administrative employee. By encapsulating the reports attribute, we ensure that it can only be accessed or modified through the provided methods.
2. Department: The AdministrativeEmployee class includes a private attribute called department, which is a string that represents the department to which the administrative employee belongs. This attribute allows you to track and store information about the employee's department.
3. getReports(): This method is a public accessor that returns the list of reports associated with the administrative employee. It provides a way to retrieve the reports for further processing or displaying.
4. addReport(report: Report): This method allows you to add a new report to the list of reports. It takes a Report object as a parameter and appends it to the existing list.
5. removeReport(report: Report): This method enables you to remove a specific report from the list of reports. It takes a Report object as a parameter and removes it from the list if it exists.
6. getDepartment(): This method is a public accessor that returns the department of the administrative employee. It allows you to retrieve the department information for the employee.
7. setDepartment(department: string): This method is a public mutator that sets the department of the administrative employee. It takes a string parameter representing the department and updates the department attribute accordingly.

بناءً على المعلومات المقدمة، هنا هي العلاقات بين الفئات:

1. الفئة موظف والفئة مبرمج:

- التوريث: الفئة مبرمج ترث من الفئة موظف. هذا يمثل علاقة "هو-نوع"، حيث يُعتبر المبرمج نوعًا من الموظفين.

2. الفئة موظف والفئة موظف إداري:

- التوريث: الفئة موظف الإداري يرث من الفئة موظف. هذا يمثل علاقة "هو-نوع"، حيث يُعتبر الموظف الإداري نوعًا من الموظفين.

3. الفئة مبرمج والفئة مشروع:

- الارتباط: الفئة مبرمج تحتوي على ارتباط مع الفئة مشروع. هذا يمثل أن المبرمج يساهم في مشروع واحد أو أكثر. الفئة مبرمج تحتوي على قائمة من كائنات الفئة مشروع.

4. الفئة موظف إداري والفئة تقرير:

- الارتباط: الفئة موظف إداري تحتوي على ارتباط مع الفئة تقرير. هذا يمثل أن الموظف الإداري يقوم بإنشاء تقرير واحد أو أكثر. الفئة موظف إداري تحتوي على قائمة من كائنات الفئة تقرير.

- التجميع/التركيب: بناءً على طبيعة العلاقة بين الموظفين الإداريين والتقارير، يمكن أن تكون العلاقة تجميعية (إذا كانت التقارير يمكن أن تكون مستقلة) أو تركيبية (إذا كانت التقارير جزءًا من دورة حياة الموظف الإداري).

سوف يوضح الرسم البياني للفئات هذه العلاقات والارتباطات بين الفئات، مشيرًا إلى العلاقات التوريث والارتباط والتجميع/التركيب وفقًا لذلك. بناءً على المعلومات المقدمة، هنا هي العلاقات بين الفئات:

1. الفئة موظف والفئة مبرمج:

- التوريث: الفئة مبرمج ترث من الفئة موظف. هذا يمثل علاقة "هو-نوع"، حيث يُعتبر المبرمج نوعًا من الموظفين.

2. الفئة موظف والفئة موظف إداري:

- التوريث: الفئة موظف الإداري يرث من الفئة موظف. هذا يمثل علاقة "هو-نوع"، حيث يُعتبر الموظف الإداري نوعًا من الموظفين.

3. الفئة مبرمج والفئة مشروع:

- الارتباط: الفئة مبرمج تحتوي على ارتباط مع الفئة مشروع. هذا يمثل أن المبرمج يساهم في مشروع واحد أو أكثر. الفئة مبرمج تحتوي على قائمة من كائنات الفئة مشروع.

4. الفئة موظف إداري والفئة تقرير:

- الارتباط: الفئة موظف إداري تحتوي على ارتباط مع الفئة تقرير. هذا يمثل أن الموظف الإداري يقوم بإنشاء تقرير واحد أو أكثر. الفئة موظف إداري تحتوي على قائمة من كائنات الفئة تقرير.

- التجميع/التركيب: بناءً على طبيعة العلاقة بين الموظفين الإداريين والتقارير، يمكن أن تكون العلاقة تجميعية (إذا كانت التقارير يمكن أن تكون مستقلة) أو تركيبية (إذا كانت التقارير جزءًا من دورة حياة الموظف الإداري).

سوف يوضح الرسم البياني للفئات هذه العلاقات والارتباطات بين الفئات، مشيرًا إلى العلاقات التوريث والارتباط والتجميع/التركيب وفقًا لذلك.

Based on the provided information, here are the relationships between the classes:

1. Employee and Programmer:

- Inheritance: The `Programmer` class inherits from the `Employee` class. This represents an "is-a" relationship, where a programmer is a type of employee.

2. Employee and Administrative Employee:

- Inheritance: The `AdministrativeEmployee` class inherits from the `Employee` class. This represents an "is-a" relationship, where an administrative employee is a type of employee.

3. Programmer and Project:

- Association: The `Programmer` class has an association with the `Project` class. This represents that a programmer contributes to one or more projects. The `Programmer` class contains a list of `Project` objects.

4. Administrative Employee and Report:

- Association: The `AdministrativeEmployee` class has an association with the `Report` class. This represents that an administrative employee creates one or more reports. The `AdministrativeEmployee` class contains a list of `Report` objects.

- Aggregation/Composition: Depending on the relationship between administrative employees and reports, it can be an aggregation (if reports can exist independently) or composition (if reports are part of the administrative employee's lifecycle).

The class diagram would showcase these relationships and associations between the classes, indicating the inheritance, association, and aggregation/composition relationships accordingly.

**Activity Diagram for Online Flight Reservation System:**

يجب أن يسمح نظام حجز الرحلات عبر الإنترنت للعملاء بالبحث عن الرحلات، وإجراء الحجز، وشراء التذكرة، واختيار المقعد. عملية شراء التذكرة يتم التحقق منها من قبل البنك.

The activity diagram for the online flight reservation system consists of several sequential activities that occur during the process. Here's a description of each activity:

1. Start Activity: This is the initial activity that marks the beginning of the reservation process.

2. Search Flights: The customer initiates a search for available flights based on their preferences, such as the departure and destination airports, date, and other criteria.

3. Display Flights: The system retrieves and displays a list of available flights that match the customer's search criteria.

4. Select Flight: The customer selects a specific flight from the displayed list.

5. Make Reservation: The customer provides necessary information, such as personal details and passenger information, to make a reservation for the selected flight.

6. Select Seat: The customer selects a seat for their reservation, which may involve choosing from available options or being assigned a seat automatically.

7. Purchase Ticket: The customer proceeds to purchase the ticket for the reserved flight. This step involves interacting with a bank to verify the payment.

8. Verify Payment: The system verifies the payment information with the bank to ensure the transaction is valid and successful.

9. Print Ticket: Once the payment is verified, the system generates and provides the customer with the ticket for the reserved flight.

10. End Activity: This marks the end of the reservation process.

It's important to note that this activity diagram provides a high-level overview of the main activities involved in the online flight reservation system. The actual system may have additional steps or variations based on specific requirements and business rules.

الرسم البياني للأنشطة لنظام حجز الرحلات عبر الإنترنت يتضمن عدة أنشطة تتم بتسلسل خلال العملية. فيما يلي وصف لكل نشاط:

1. بدء النشاط: هذا هو النشاط الأولي الذي يشير إلى بداية عملية الحجز.

2. البحث عن رحلات: يقوم العميل بالبحث عن الرحلات المتاحة استنادًا إلى تفضيلاته، مثل مطار الانطلاق والوجهة، والتاريخ، ومعايير أخرى.

3. عرض الرحلات: يقوم النظام باسترجاع وعرض قائمة بالرحلات المتاحة التي تتطابق مع معايير البحث التي حددها العميل.

4. اختيار الرحلة: يقوم العميل بتحديد رحلة محددة من القائمة المعروضة.

5. إجراء الحجز: يقدم العميل المعلومات اللازمة، مثل التفاصيل الشخصية ومعلومات المسافرين، لإجراء الحجز للرحلة المحددة.

6. اختيار المقعد: يقوم العميل بتحديد مقعد لحجزه، والذي قد يتضمن اختيار من الخيارات المتاحة أو تعيين مقعد تلقائيًا.

7. شراء التذكرة: يتابع العميل عملية شراء التذكرة للرحلة المحجوزة. يتضمن هذا الخطوة التفاعل مع البنك للتحقق من الدفع.

8. التحقق من الدفع: يقوم النظام بالتحقق من معلومات الدفع مع البنك لضمان صحة ونجاح العملية.

9. طباعة التذكرة: بمجرد التحقق من الدفع، يقوم النظام بإنشاء التذكرة وتوفيرها للعميل للرحلة المحجوزة.

10. انتهاء النشاط: يشير هذا إلى انتهاء عملية الحجز.

من المهم أن نلاحظ أن هذا الرسم البياني للأنشطة يوفر نظرة عامة عن الأنشطة الرئيسية المشتركة في نظام حجز الرحلات عبر الإنترنت. قد يحتوي النظام الفعلي على خطوات إضافية أو اختلافات بناءً على المتطلبات وقواعد العمل المحددة.

In an online flight reservation system, several **use case diagrams** can be created to represent different functionalities and interactions within the system. Here are some example use case diagrams that can be included in the online flight reservation system:

User Registration and Login Use Case Diagram: This diagram would include use cases such as "Register User," "Login," and "Logout." It represents the interaction between users and the system for account creation, authentication, and session management.

Search and Select Flights Use Case Diagram: This diagram would include use cases such as "Search Flights," "View Flight Details," and "Select Flight." It represents the interaction between users and the system for searching available flights, viewing flight details, and selecting a preferred flight.

3. Make Reservation Use Case Diagram: This diagram would include use cases such as "Enter Passenger Details," "Choose Seat," and "Confirm Reservation." It represents the interaction between users and the system for entering passenger information, selecting seats, and finalizing the reservation.

4. Payment and Ticketing Use Case Diagram: This diagram would include use cases such as "Enter Payment Details," "Verify Payment," and "Issue Ticket." It represents the interaction between users, the system, and the payment gateway or bank for processing payments, verifying transactions, and generating tickets.

5. Manage Reservations Use Case Diagram: This diagram would include use cases such as "View Reservations," "Modify Reservation," and "Cancel Reservation." It represents the interaction between users and the system for managing existing reservations, including viewing reservation details, making modifications, and canceling bookings.

6. Admin Management Use Case Diagram: This diagram would include use cases related to administrative functionalities such as "Manage Flights," "Manage Users," and "Generate Reports." It represents the interaction between administrators and the system for managing flight information, user accounts, and generating reports.

These are just some examples of the use case diagrams that can be created for an online flight reservation system. The specific use cases and interactions may vary based on the requirements and features of the system.

بالطبع! يُمكن إنشاء العديد من رسومات حالات الاستخدام (Use Case Diagrams) لنظام حجز الرحلات عبر الإنترنت لتمثيل وظائف وتفاعلات مختلفة داخل النظام. هنا بعض أمثلة لرسومات حالات الاستخدام التي يُمكن تضمينها في نظام حجز الرحلات عبر الإنترنت:

1. رسم حالات الاستخدام لتسجيل المستخدم وتسجيل الدخول: يحتوي هذا الرسم على حالات استخدام مثل "تسجيل المستخدم" و "تسجيل الدخول" و "تسجيل الخروج". يمثل التفاعل بين المستخدمين والنظام لإنشاء حساب جديد والمصادقة وإدارة الجلسات.

2. رسم حالات الاستخدام للبحث عن الرحلات واختيارها: يحتوي هذا الرسم على حالات استخدام مثل "البحث عن الرحلات" و "عرض تفاصيل الرحلة" و "اختيار الرحلة". يمثل التفاعل بين المستخدمين والنظام للبحث عن الرحلات المتاحة وعرض تفاصيل الرحلة واختيار الرحلة المفضلة.

3. رسم حالات الاستخدام لإجراء الحجز: يحتوي هذا الرسم على حالات استخدام مثل "إدخال تفاصيل المسافرين" و "اختيار المقعد" و "تأكيد الحجز". يمثل التفاعل بين المستخدمين والنظام لإدخال معلومات المسافرين واختيار المقاعد وتأكيد الحجز.

4. رسم حالات الاستخدام للدفع وإصدار التذكرة: يحتوي هذا الرسم على حالات استخدام مثل "إدخال تفاصيل الدفع" و "التحقق من الدفع" و "إصدار التذكرة". يمثل التفاعل بين المستخدمين والنظام وبوابة الدفع أو البنك لمعالجة الدفعات والتحقق من المعاملات وإصدار التذاكر.

5. رسم حالات الاستخدام لإدارة الحجوزات: يحتوي هذا الرسم على حالات استخدام مثل "عرض الحجوزات" و "تعديل الحجز" و "إلغاء الحجز". يمثل التفاعل بين المستخدمين والنظام لإدارة الحجوزات الحالية، بما في ذلك عرض تفاصيل الحجز وإجراء التعديلات وإلغاء الحجوزات.

6- رسم حالات الاستخدام لإدارة المشرف: يحتوي هذا الرسم على حالات استخدام ذات صلة بالوظائف الإدارية مثل "إأعتذر، ولكن لم يتم توفير إكمال الجملة في الاستفسار السابق. يرجى توفير المزيد من المعلومات أو إعادة صياغة السؤال بشكل صحيح حتى أتمكن من مساعدتك بشكل أفضل.

Based on the requirements you provided, the following are the names of the use case diagrams that can be created for the online flight reservation system:

1. User Registration and Login Use Case Diagram: This diagram represents the use cases related to user registration, login, and logout functionalities.

2. Search Flights Use Case Diagram: This diagram represents the use cases for searching and retrieving available flights based on user-specified criteria.

3. Make Reservation Use Case Diagram: This diagram represents the use cases for making a flight reservation, including entering passenger details and selecting a flight.

4. Purchase Ticket Use Case Diagram: This diagram represents the use cases for purchasing a ticket, including providing payment details and verifying the transaction with the bank.

5. Select Seat Use Case Diagram: This diagram represents the use cases for selecting a seat during the reservation process.

These use case diagrams illustrate the various functionalities and interactions within the online flight reservation system, including user management, flight search, reservation creation, ticket purchase, and seat selection.

As a text-based AI model, I'm unable to draw diagrams directly. However, I can describe the activity diagram for the "User Registration and Login" use case in the online flight reservation system.

The "User Registration and Login" activity diagram would typically consist of the following activities and transitions:

1. Start: This is the initial activity of the diagram.

2. User Registration: This activity represents the process of user registration. It involves gathering user information such as name, email, and password.

3. Validate User Information: This activity verifies the user-provided information for registration, ensuring that all required fields are filled correctly.

4. Create User Account: This activity creates a new user account in the system, storing the user's information securely.

5. Login: This activity represents the process of user login. It involves entering the registered email and password.

6. Authenticate User: This activity verifies the entered login credentials against the stored user account information.

7. Validate Authentication: This activity checks whether the authentication is successful or not.

8. Grant Access: If the authentication is successful, this activity grants access to the system, allowing the user to proceed to the next steps.

9. End: This activity indicates the completion of the "User Registration and Login" process.

The activity diagram would show the flow of activities and decisions involved in the user registration and login process. Arrows or transitions would connect the activities, indicating the sequence of steps. Decision points can be represented using diamond shapes, where different paths can be taken based on conditions or validation results.

Please note that the actual layout and design of the activity diagram may vary based on individual preferences and the specific tool or software used to create the diagram.

As a text-based AI, I'm unable to draw diagrams directly. However, I can describe the activity diagram for the "Search Flights" use case in the online flight reservation system.

The "Search Flights" activity diagram would typically consist of the following activities and transitions:

1. Start: This is the initial activity of the diagram.

2. Enter Search Criteria: This activity represents the user entering search criteria, such as departure location, arrival location, travel dates, and any other relevant filters.

3. Validate Search Criteria: This activity validates the entered search criteria, checking for any missing or invalid information.

4. Retrieve Flights: This activity retrieves the available flights from the system's database based on the provided search criteria.

5. Display Flight Results: This activity displays the retrieved flight results to the user, showing details such as flight numbers, departure times, arrival times, and prices.

6. Select Flight: This activity represents the user selecting a specific flight from the displayed results.

7. Confirm Selection: This activity confirms the user's selection and proceeds to the next steps.

8. End: This activity indicates the completion of the "Search Flights" process.

The activity diagram would show the flow of activities and decisions involved in the flight search process. Arrows or transitions would connect the activities, indicating the sequence of steps. Decision points can be represented using diamond shapes, where different paths can be taken based on conditions or validation results.

Please note that the actual layout and design of the activity diagram may vary based on individual preferences and the specific tool or software used to create the diagram.

As a text-based AI, I'm unable to draw diagrams directly. However, I can describe the activity diagram for the "Make Reservation" use case in the online flight reservation system.

The "Make Reservation" activity diagram would typically consist of the following activities and transitions:

1. Start: This is the initial activity of the diagram.

2. Select Flight: This activity represents the user selecting a specific flight from the available options.

3. Enter Passenger Details: This activity involves the user entering the necessary passenger details, such as names, contact information, and any additional requirements.

4. Select Seat: This activity allows the user to choose a seat from the available seating options on the selected flight.

5. Validate Passenger Details: This activity validates the entered passenger details, ensuring that all required fields are filled correctly.

6. Confirm Reservation: This activity confirms the user's reservation, taking into account the selected flight and seat, as well as the entered passenger details.

7. Check Availability: This activity checks the availability of the selected flight and seat to ensure that it is still available at the time of reservation.

8. Make Payment: This activity involves the user making the necessary payment for the reservation, either by providing credit card information or using an alternative payment method.

9. Validate Payment: This activity verifies the payment details and ensures that the transaction is successful.

10. Generate Confirmation: This activity generates a reservation confirmation number or ticket for the user, indicating the successful completion of the reservation process.

11. End: This activity indicates the completion of the "Make Reservation" process.

The activity diagram would show the flow of activities and decisions involved in the reservation process. Arrows or transitions would connect the activities, indicating the sequence of steps. Decision points can be represented using diamond shapes, where different paths can be taken based on conditions or validation results.

Please note that the actual layout and design of the activity diagram may vary based on individual preferences and the specific tool or software used to create the diagram.

As a text-based AI, I'm unable to draw diagrams directly. However, I can describe the activity diagram for the "Purchase Ticket" use case in the online flight reservation system.

The "Purchase Ticket" activity diagram would typically consist of the following activities and transitions:

1. Start: This is the initial activity of the diagram.

2. Select Flight: This activity represents the user selecting a specific flight for ticket purchase.

3. Enter Passenger Details: This activity involves the user entering the necessary passenger details, such as names, contact information, and any additional requirements.

4. Select Seat: This activity allows the user to choose a seat from the available seating options on the selected flight.

5. Validate Passenger Details: This activity validates the entered passenger details, ensuring that all required fields are filled correctly.

6. Confirm Reservation: This activity confirms the user's reservation, taking into account the selected flight, seat, and passenger details.

7. Make Payment: This activity involves the user making the necessary payment for the ticket purchase, either by providing credit card information or using an alternative payment method.

8. Verify Payment: This activity verifies the payment details and initiates the payment transaction.

9. Process Payment: This activity processes the payment transaction, interacting with the bank or payment gateway to complete the payment.

10. Receive Payment Confirmation: This activity receives the confirmation of the successful payment transaction.

11. Generate Ticket: This activity generates the ticket for the user, including the ticket number, flight details, passenger information, and seat assignment.

12. End: This activity indicates the completion of the "Purchase Ticket" process.

The activity diagram would show the flow of activities and decisions involved in the ticket purchase process. Arrows or transitions would connect the activities, indicating the sequence of steps. Decision points can be represented using diamond shapes, where different paths can be taken based on conditions or validation results.

Please note that the actual layout and design of the activity diagram may vary based on individual preferences and the specific tool or software used to create the diagram.

As a text-based AI, I'm unable to draw diagrams directly. However, I can describe the activity diagram for the "Purchase Ticket" use case in the online flight reservation system.

The "Purchase Ticket" activity diagram would typically consist of the following activities and transitions:

1. Start: This is the initial activity of the diagram.

2. Select Flight: This activity represents the user selecting a specific flight for ticket purchase.

3. Enter Passenger Details: This activity involves the user entering the necessary passenger details, such as names, contact information, and any additional requirements.

4. Select Seat: This activity allows the user to choose a seat from the available seating options on the selected flight.

5. Validate Passenger Details: This activity validates the entered passenger details, ensuring that all required fields are filled correctly.

6. Confirm Reservation: This activity confirms the user's reservation, taking into account the selected flight, seat, and passenger details.

7. Make Payment: This activity involves the user making the necessary payment for the ticket purchase, either by providing credit card information or using an alternative payment method.

8. Verify Payment: This activity verifies the payment details and initiates the payment transaction.

9. Process Payment: This activity processes the payment transaction, interacting with the bank or payment gateway to complete the payment.

10. Receive Payment Confirmation: This activity receives the confirmation of the successful payment transaction.

11. Generate Ticket: This activity generates the ticket for the user, including the ticket number, flight details, passenger information, and seat assignment.

12. End: This activity indicates the completion of the "Purchase Ticket" process.

The activity diagram would show the flow of activities and decisions involved in the ticket purchase process. Arrows or transitions would connect the activities, indicating the sequence of steps. Decision points can be represented using diamond shapes, where different paths can be taken based on conditions or validation results.

Please note that the actual layout and design of the activity diagram may vary based on individual preferences and the specific tool or software used to create the diagram.

As a text-based AI, I'm unable to draw diagrams directly. However, I can describe the activity diagram for the "Select Seat" use case in the online flight reservation system.

The "Select Seat" activity diagram would typically consist of the following activities and transitions:

1. Start: This is the initial activity of the diagram.

2. Display Seat Map: This activity involves displaying the seat map for the selected flight, showing the available and occupied seats.

3. Select Seat: This activity represents the user selecting a specific seat from the available options on the seat map.

4. Validate Seat Selection: This activity validates the selected seat, ensuring that it is still available and not already occupied by another passenger.

5. Confirm Seat Selection: This activity confirms the user's seat selection and proceeds to the next steps.

6. End: This activity indicates the completion of the "Select Seat" process.

The activity diagram would show the flow of activities and decisions involved in the seat selection process. Arrows or transitions would connect the activities, indicating the sequence of steps. Decision points can be represented using diamond shapes, where different paths can be taken based on conditions or validation results.

Please note that the actual layout and design of the activity diagram may vary based on individual preferences and the specific tool or software used to create the diagram.