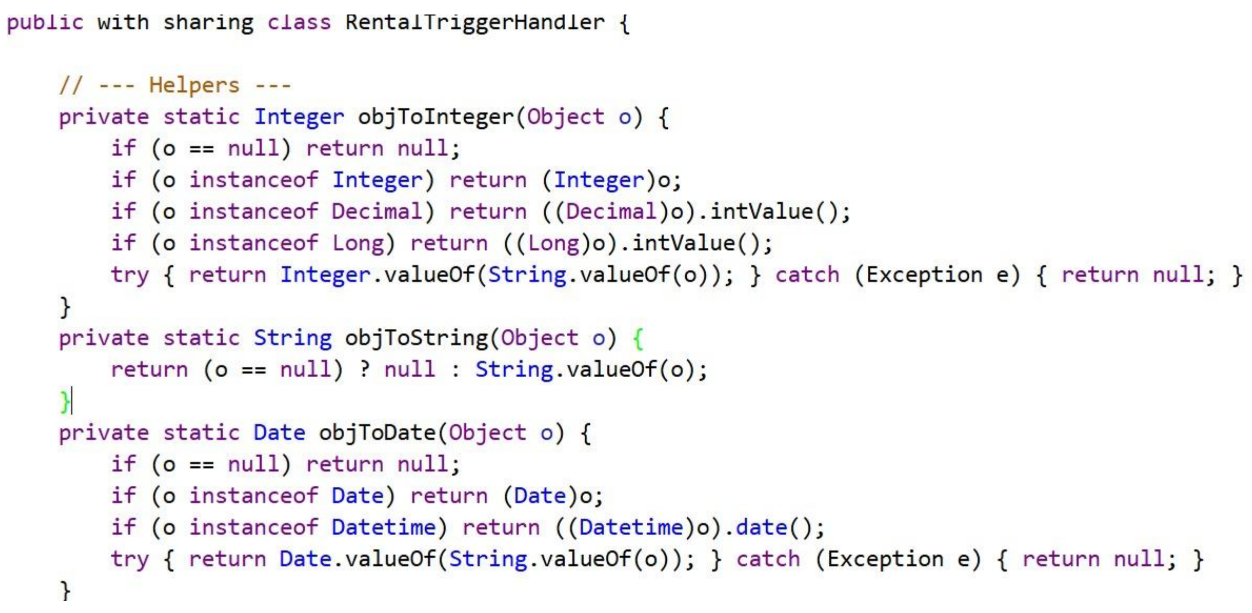
Phase 5: Testing, UAT, & Deployment

This phase involves comprehensive testing and user acceptance testing (UAT) of the newly developed features, followed by deployment to the production environment. The goal is to ensure all functionalities work as designed, meet academic and business requirements, and are ready for end-users.

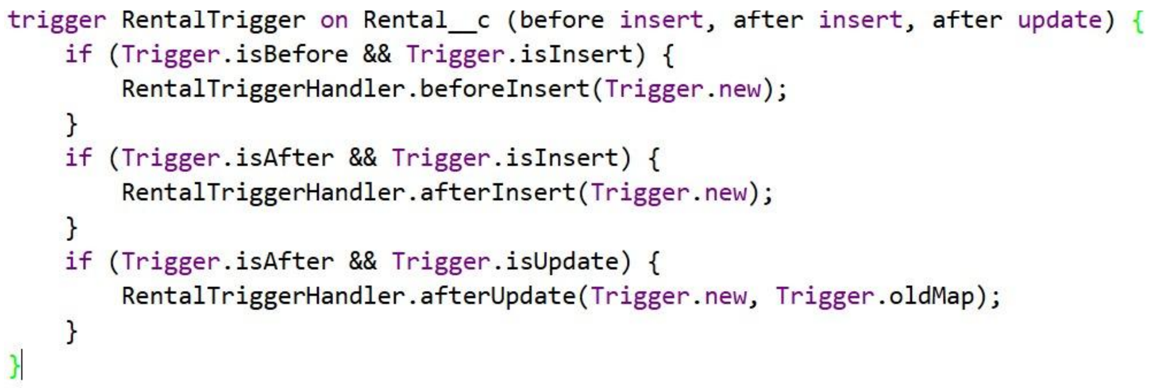
Testing & Quality Assurance

- **Unit Testing:** Individual Apex classes and triggers, such as the RentalTriggerHandler and its helper methods, are tested to ensure they function correctly under various conditions. The code snippets show a test class named RentalTriggerHandlerFullTest with a utility method to create Book__c records, suggesting that unit tests are being written to validate the trigger's logic . These tests would cover scenarios like:
  - Validating the Rent_Date__c and Due_Date__c fields .
  - Checking for available copies before a rental is created .
  - Handling the Rented status to prevent rentals of unavailable books .



```
public with sharing class RentalTriggerHandler {

    // --- Helpers ---
    private static Integer objToInteger(Object o) {
        if (o == null) return null;
        if (o instanceof Integer) return (Integer)o;
        if (o instanceof Decimal) return ((Decimal)o).intValue();
        if (o instanceof Long) return ((Long)o).intValue();
        try { return Integer.valueOf(String.valueOf(o)); } catch (Exception e) { return null; }
    }
    private static String objToString(Object o) {
        return (o == null) ? null : String.valueOf(o);
    }
    private static Date objToDate(Object o) {
        if (o == null) return null;
        if (o instanceof Date) return (Date)o;
        if (o instanceof Datetime) return ((Datetime)o).date();
        try { return Date.valueOf(String.valueOf(o)); } catch (Exception e) { return null; }
    }
```

- **Integration Testing:** Testing how the Rental__c and Book__c objects interact with other parts of the system and with each other. This includes verifying that the trigger correctly updates the number of available books and that the user's role and profile permissions are enforced.

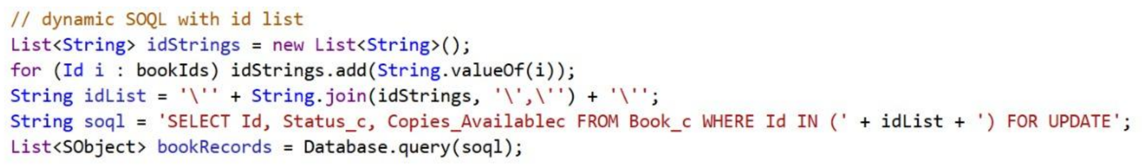


```
trigger RentalTrigger on Rental__c (before insert, after insert, after update) {
    if (Trigger.isBefore && Trigger.isInsert) {
        RentalTriggerHandler.beforeInsert(Trigger.new);
    }
    if (Trigger.isAfter && Trigger.isInsert) {
        RentalTriggerHandler.afterInsert(Trigger.new);
    }
    if (Trigger.isAfter && Trigger.isUpdate) {
        RentalTriggerHandler.afterUpdate(Trigger.new, Trigger.oldMap);
    }
}
```

- **System Testing:** End-to-end testing of the entire rental process, from a student user requesting a book to a teacher or admin managing the rental record. This would use the roles and profiles defined in Phase 2.

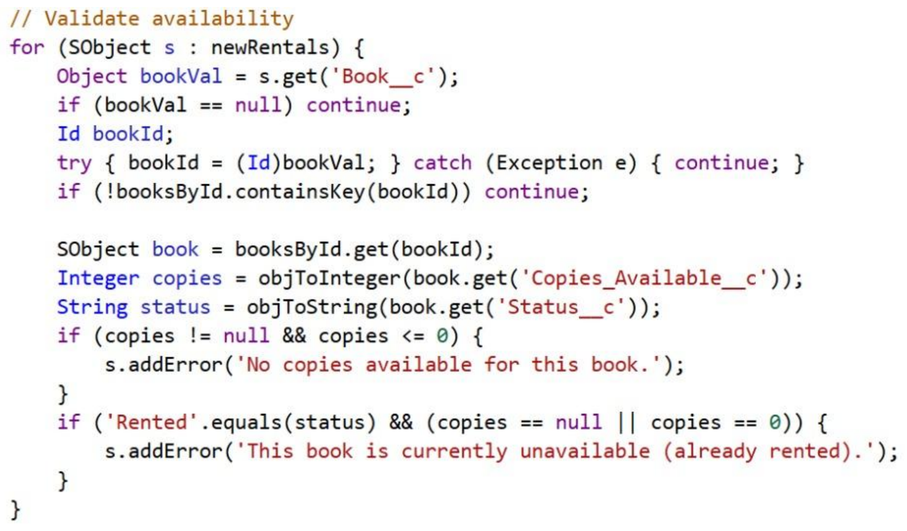
User Acceptance Testing (UAT)

- **UAT Scope:** Key stakeholders, including academic staff and students, will test the new functionalities in a sandbox environment to ensure the system meets their needs.



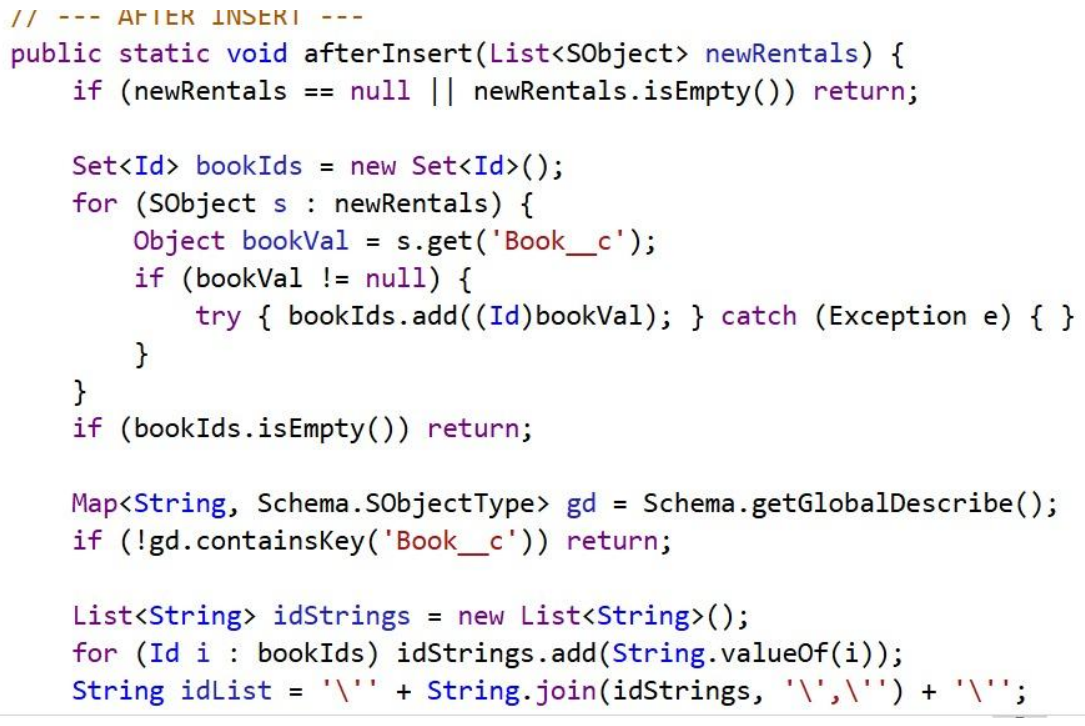
```
// dynamic SOQL with id list
List<String> idStrings = new List<String>();
for (Id i : bookIds) idStrings.add(String.valueOf(i));
String idList = '\'' + String.join(idStrings, '\',\'') + '\'';
String soql = 'SELECT Id, Status_c, Copies_Availablec FROM Book_c WHERE Id IN (' + idList + ') FOR UPDATE';
List<SObject> bookRecords = Database.query(soql);
```

- **Test Scenarios:**
  - A student user successfully rents an available book.
  - An admin user manually updates a rental record.
  - An attempt is made to rent a book that has no copies available, and the correct error message is displayed, as seen in the Validate availability code .
  - A guardian user attempts to view a Submission__c record, verifying that sharing rules are properly restricting access.

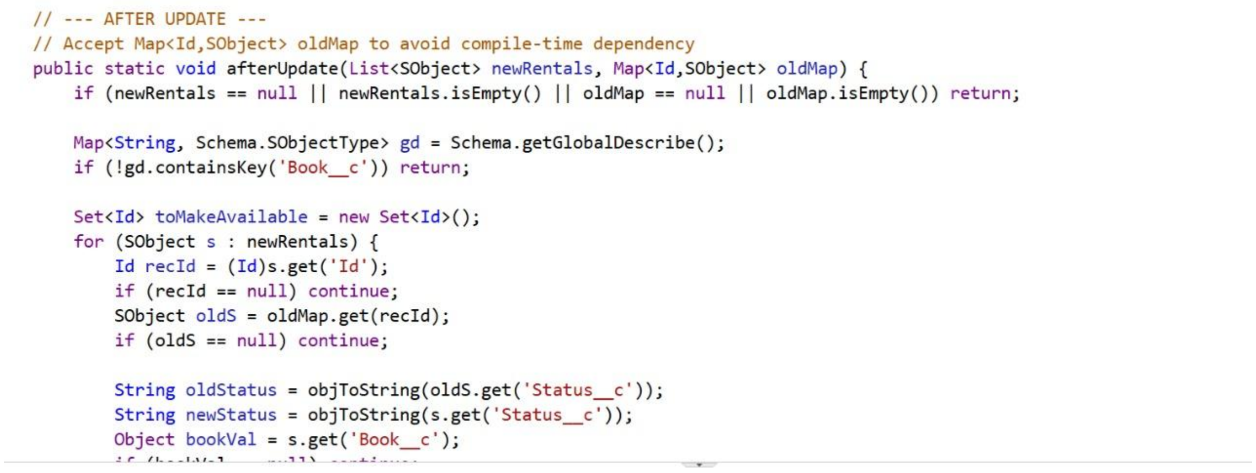


```
// Validate availability
for (SObject s : newRentals) {
    Object bookVal = s.get('Book__c');
    if (bookVal == null) continue;
    Id bookId;
    try { bookId = (Id)bookVal; } catch (Exception e) { continue; }
    if (!booksById.containsKey(bookId)) continue;

    SObject book = booksById.get(bookId);
    Integer copies = objToInteger(book.get('Copies_Available__c'));
    String status = objToString(book.get('Status__c'));
    if (copies != null && copies <= 0) {
        s.addError('No copies available for this book.');
    }
    if ('Rented'.equals(status) && (copies == null || copies == 0)) {
        s.addError('This book is currently unavailable (already rented).');
    }
}
```

- **Feedback & Iteration:** UAT feedback is collected, and any necessary adjustments or bug fixes are made before the final deployment.



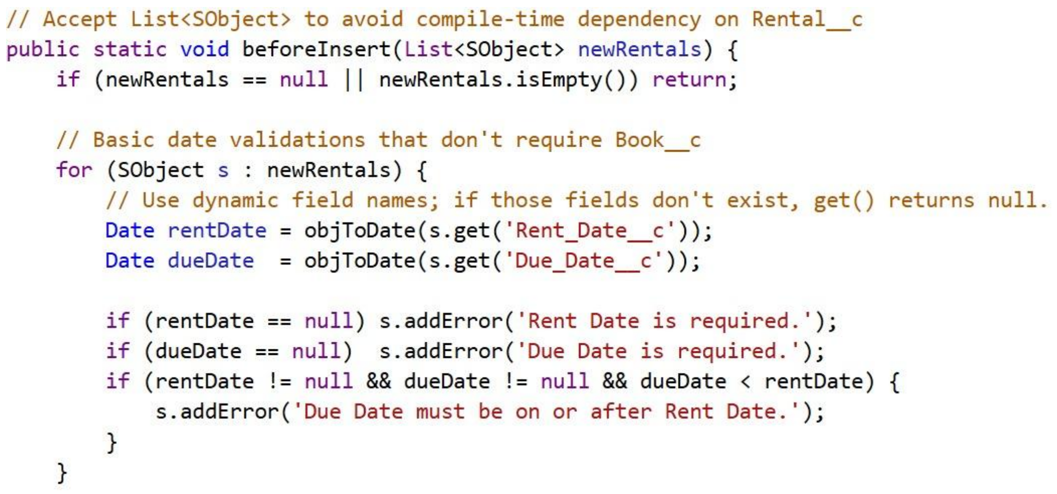
```
// --- AFTER INSERT ---
public static void afterInsert(List<SObject> newRentals) {
    if (newRentals == null || newRentals.isEmpty()) return;

    Set<Id> bookIds = new Set<Id>();
    for (SObject s : newRentals) {
        Object bookVal = s.get('Book__c');
        if (bookVal != null) {
            try { bookIds.add((Id)bookVal); } catch (Exception e) { }
        }
    }
    if (bookIds.isEmpty()) return;

    Map<String, Schema.SObjectType> gd = Schema.getGlobalDescribe();
    if (!gd.containsKey('Book__c')) return;

    List<String> idStrings = new List<String>();
    for (Id i : bookIds) idStrings.add(String.valueOf(i));
    String idList = '\'' + String.join(idStrings, '\',\'') + '\'';
```



```
// --- AFTER UPDATE ---
// Accept Map<Id,SObject> oldMap to avoid compile-time dependency
public static void afterUpdate(List<SObject> newRentals, Map<Id,SObject> oldMap) {
    if (newRentals == null || newRentals.isEmpty() || oldMap == null || oldMap.isEmpty()) return;

    Map<String, Schema.SObjectType> gd = Schema.getGlobalDescribe();
    if (!gd.containsKey('Book__c')) return;

    Set<Id> toMakeAvailable = new Set<Id>();
    for (SObject s : newRentals) {
        Id recId = (Id)s.get('Id');
        if (recId == null) continue;
        SObject oldS = oldMap.get(recId);
        if (oldS == null) continue;

        String oldStatus = objToString(oldS.get('Status__c'));
        String newStatus = objToString(s.get('Status__c'));
        Object bookVal = s.get('Book__c');
```

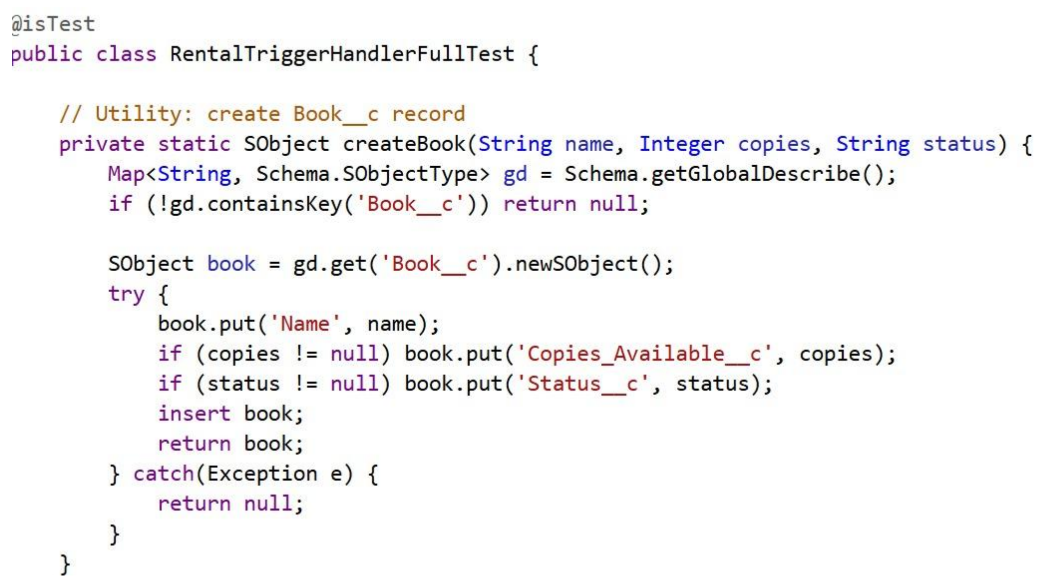
Deployment

- **Deployment Strategy:** A change set or a more advanced tool like Salesforce DX is used to migrate the tested components from the sandbox to the production environment. This includes the custom objects (Book__c, Rental__c), Apex classes, triggers, and any related custom fields or user interface elements.

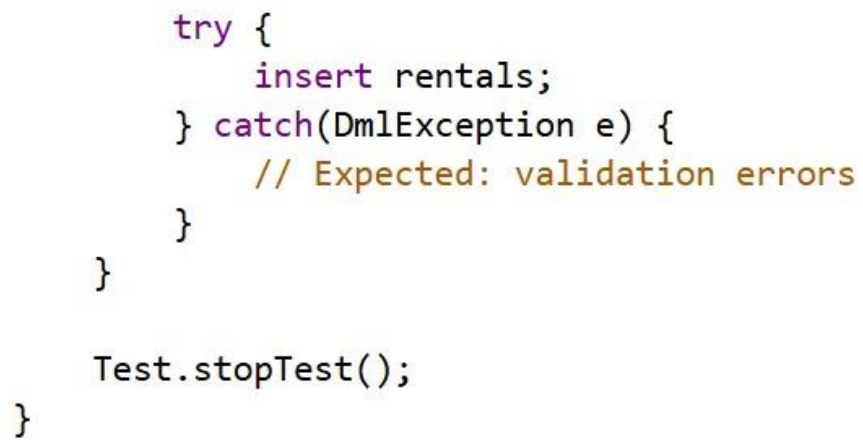


```
// Accept List<SObject> to avoid compile-time dependency on Rental__c
public static void beforeInsert(List<SObject> newRentals) {
    if (newRentals == null || newRentals.isEmpty()) return;

    // Basic date validations that don't require Book__c
    for (SObject s : newRentals) {
        // Use dynamic field names; if those fields don't exist, get() returns null.
        Date rentDate = objToDate(s.get('Rent_Date__c'));
        Date dueDate  = objToDate(s.get('Due_Date__c'));

        if (rentDate == null) s.addError('Rent Date is required.');
        if (dueDate == null)  s.addError('Due Date is required.');
        if (rentDate != null && dueDate != null && dueDate < rentDate) {
            s.addError('Due Date must be on or after Rent Date.');
        }
    }
```

- **Post-Deployment Validation:** A final check to ensure all components are deployed correctly and are functioning as expected in the live environment.



```
@isTest
public class RentalTriggerHandlerFullTest {

    // Utility: create Book__c record
    private static SObject createBook(String name, Integer copies, String status) {
        Map<String, Schema.SObjectType> gd = Schema.getGlobalDescribe();
        if (!gd.containsKey('Book__c')) return null;

        SObject book = gd.get('Book__c').newSObject();
        try {
            book.put('Name', name);
            if (copies != null) book.put('Copies_Available__c', copies);
            if (status != null) book.put('Status__c', status);
            insert book;
            return book;
        } catch(Exception e) {
            return null;
        }
    }
```



```
            try {
                insert rentals;
            } catch(DmlException e) {
                // Expected: validation errors
            }
        }

        Test.stopTest();
    }
```

- **Go-Live:** The new book rental system is made available to all users. A communication plan would be executed to inform students and faculty about the new functionality.

Outcomes of Phase 5

This phase results in a fully tested and deployed book rental system within the Salesforce environment, providing a functional solution for managing library resources. The successful completion of this phase ensures a smooth transition to the new system, minimizes disruption, and validates that the development work aligns with the institution's academic and operational needs.