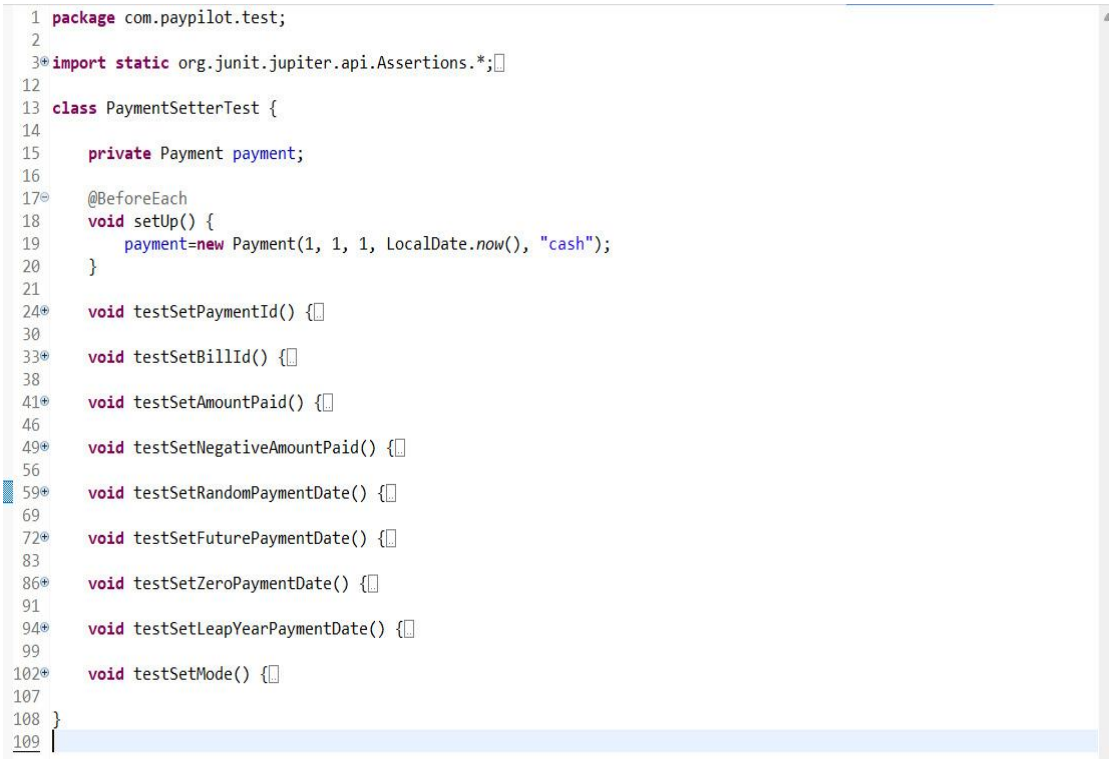
NATWEST BOOTCAMP

Group Report: Group 2
Hackathon 1 & Hackathon 2

Report HACKATHON 1: 28th July, 2025

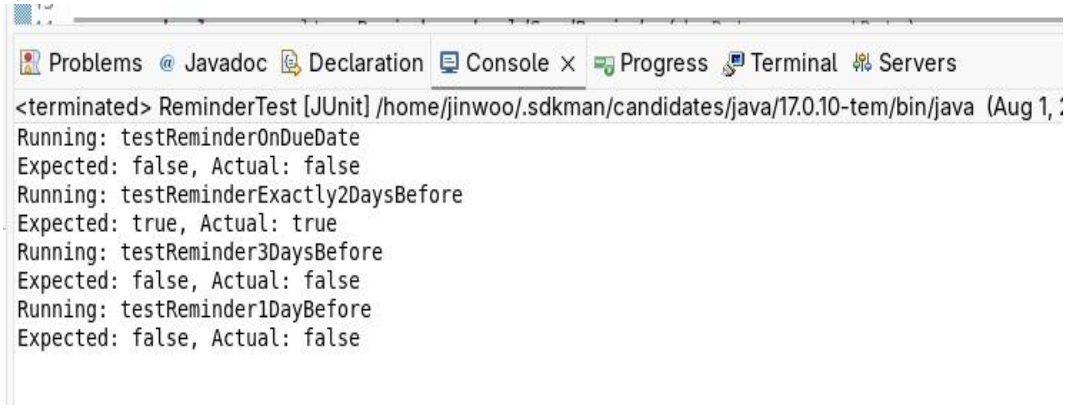
1. **TASK: Write a Junit test for Bill and Payment both for setters, date parsing with different modes of edge cases like zero amount etc**



```
1  package com.paypilot.test;
2
3  import static org.junit.jupiter.api.Assertions.*;
12
13 class PaymentSetterTest {
14
15     private Payment payment;
16
17     @BeforeEach
18     void setUp() {
19         payment=new Payment(1, 1, 1, LocalDate.now(), "cash");
20     }
21
24     void testSetPaymentId() {
30
33     void testSetBillId() {
38
41     void testSetAmountPaid() {
46
49     void testSetNegativeAmountPaid() {
56
59     void testSetRandomPaymentDate() {
69
72     void testSetFuturePaymentDate() {
83
86     void testSetZeroPaymentDate() {
91
94     void testSetLeapYearPaymentDate() {
99
102    void testSetMode() {
107
108 }
109
```

```java
package com.paypilot.test;

import static org.junit.jupiter.api.Assertions.*;

class BillSetterTest {

    private Bill bill;

    @BeforeEach
    void setUp() {
        //create new bill object
        bill=new Bill(0, 0, "a", "a", LocalDate.now(), 0, false);
    }

    void testSetUserId() {

    void testSetBillId() {

    void testSetBillName() {

    void testSetCategory() {

    void testSetRandomDueDate() {

    void testSetFutureDueDate() {

    void testSetZeroDueDate() {

    void testSetLeapYearDueDate() {

    void testSetAmount() {

    void testSetRecurring() {


}
```

2. **TASK: Write Test cases of remained generation login (Should trigger 2 days before due date)**



Code: *ReminderTest.java*

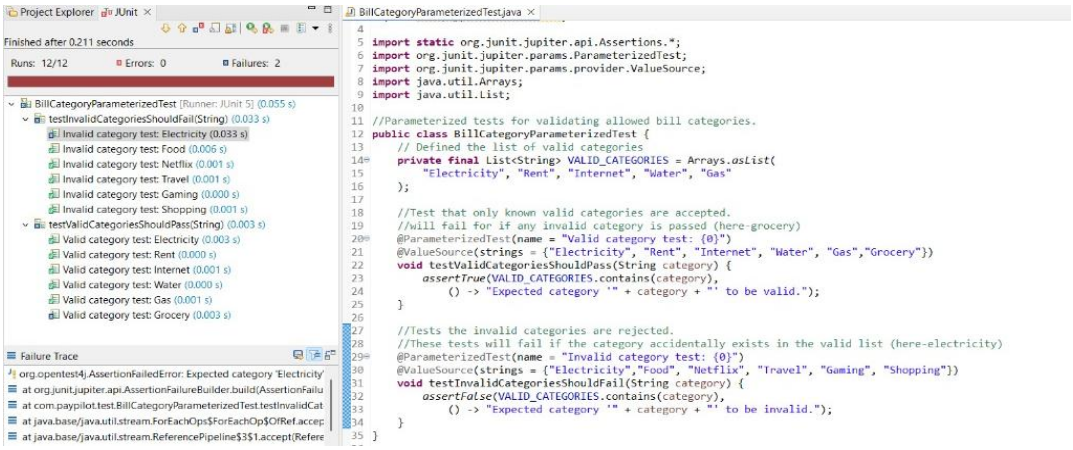
```java
package paypilot;
import static org.junit.jupiter.api.Assertions.*;
import java.time.LocalDate;
import org.junit.jupiter.api.Test;
public class ReminderTest {
    @Test
    public void testReminderExactly2DaysBefore() {
        System.out.println("Running: testReminderExactly2DaysBefore");
        LocalDate dueDate = LocalDate.of(2025, 8, 10);
        LocalDate currentDate = LocalDate.of(2025, 8, 8);
        boolean result = Reminder.shouldSendReminder(dueDate, currentDate);
        System.out.println("Expected: true, Actual: " + result);
        assertTrue(result, "Reminder should be sent exactly 2 days before due date");
    }
    @Test
    public void testReminder1DayBefore() {
        System.out.println("Running: testReminder1DayBefore");
        LocalDate dueDate = LocalDate.of(2025, 8, 10);
        LocalDate currentDate = LocalDate.of(2025, 8, 9);
        boolean result = Reminder.shouldSendReminder(dueDate, currentDate);
        System.out.println("Expected: false, Actual: " + result);
        assertFalse(result, "Reminder should NOT be sent 1 day before");
    }
    @Test
    public void testReminder3DaysBefore() {
        System.out.println("Running: testReminder3DaysBefore");
        LocalDate dueDate = LocalDate.of(2025, 8, 10);
        LocalDate currentDate = LocalDate.of(2025, 8, 7);
        boolean result = Reminder.shouldSendReminder(dueDate, currentDate);
        System.out.println("Expected: false, Actual: " + result);
        assertFalse(result, "Reminder should NOT be sent 3 days before");
    }
    @Test
    public void testReminderOnDueDate() {
        System.out.println("Running: testReminderOnDueDate");
        LocalDate dueDate = LocalDate.of(2025, 8, 10);
        LocalDate currentDate = LocalDate.of(2025, 8, 10);
        boolean result = Reminder.shouldSendReminder(dueDate, currentDate);
        System.out.println("Expected: false, Actual: " + result);
        assertFalse(result, "Reminder should NOT be sent on the due date itself");
    }
}
```

3. **TASK: Implement a function to validate if payment date is not before the due date, throw custom exception InvalidPaymentDateException.**

```java
    @Test
    public  void testValidateDate() {
        assertThrows(InvalidPaymentDateException.class,
                     () -> PaymentService.validatePaymentDate(bill, payment1)
        );
    }
}
```

```java
package com.paypilot.util;

import com.paypilot.exception.InvalidPaymentDateException;
import com.paypilot.model.Bill;
import com.paypilot.model.Payment;

public class PaymentDateValidator {
        public static void validatePaymentDate(Bill bill, Payment payment) throws InvalidPaymentDateException{
                if (payment.getPaymentDate().isBefore(bill.getDueDate())) {
            throw new InvalidPaymentDateException("Payment date cannot be before the due date.");
        } else {
            System.out.println("Payment date is valid." + payment.getPaymentDate());
        }
        }
}
```

4. **TASK: Write parameterized tests to check valid bill categories.**



5. **TASK: Write a method to check if a bill is recurring and autogenerate next month's bill,findAllBills()**

```java
// Method to find and return all bills
public List<Bill> findAllBills() {
        return new ArrayList<>(billList); //Returning a copy of Bill Lists to protect internal list
}

// Function to check if a bill is recurring
public static boolean isBillRecurring(Bill bill) {
  return bill != null && bill.isRecurring();
}

// Function to avoid generating a recurring bill if it already exists
public static boolean hasRecurringBillForNextMonth(Bill originalBill, List<Bill> allBills) {
        LocalDate nextDueDate = originalBill.getDueDate().plusMonths(1);

        return allBills.stream().anyMatch(existing ->
            existing.getUserId() == originalBill.getUserId() &&
            existing.getBillName().equalsIgnoreCase(originalBill.getBillName()) &&
            existing.getCategory().equalsIgnoreCase(originalBill.getCategory()) &&
            existing.getDueDate().equals(nextDueDate)
        );
}
```

```java
// Function to check if a bill is recurring and autogenerating next months's bill
public static List<Bill> checkAndGenerateRecurringBills(List<Bill> bills)
{
      List<Bill> newBills = new ArrayList<>();
      Set<Integer> existingIds = new HashSet<>();
      LocalDate currentDate = LocalDate.now();
      //Adding existing IDs to the set
      for (Bill bill : bills)
      {
          existingIds.add(bill.getBillId());
      }
      for (Bill bill : bills)
      {
          if (bill.isRecurring() && bill.getDueDate().isBefore(currentDate))
          {
              // Generating a unique ID that doesn't clash
              int newId;
              do {
                  Random random = new Random();

                  newId = random.nextInt(Integer.MAX_VALUE); // Ensures a non-negative int
              } while (existingIds.contains(newId));
              existingIds.add(newId);
              // Generating Date for the next month's bill
              LocalDate nextDueDate = bill.getDueDate().plusMonths(1);
               //Checking if there is a bill already generated for next month
              if (!hasRecurringBillForNextMonth(bill, bills))
              {
                  // Creating bill for the next month
                  Bill nextMonthBill = new Bill(
                      bill.getUserId(),
                      newId,
                      bill.getBillName(),
                      bill.getCategory(),
                      nextDueDate,
                      bill.getAmount(),
                      true
                  );
                  newBills.add(nextMonthBill);
              }
          }
      }
      return newBills;
}
```

6. **TASK: Write an assert for throwing custom exceptions when invalid dates are given.**

```java
package com.paypilot.test;
import org.junit.jupiter.api.Test;
public class DateUtilTest {

    //Valid date input should return correct LocalDate
    @Test
    void testParseValidDate() {
        String validDate = "28-07-2025";
        LocalDate expected = LocalDate.of(2025, 7, 28);

        LocalDate result = DateUtil.parse(validDate);
        assertEquals(expected, result, "Parsed date should match expected LocalDate");
    }

    //Invalid format (e.g. wrong order "yyyy-MM-dd") should throw exception
    @Test
    void testParseInvalidFormatThrowsException() {
        String invalidDate = "2025-07-28";

        Exception exception = assertThrows(IllegalArgumentException.class, () -> {
            DateUtil.parse(invalidDate);
        });

        assertTrue(exception.getMessage().contains("Invalid date format"), "Should contain 'Invalid date format'");
    }

    //Invalid date string like "invalid-date" should throw exception
    @Test
    void testParseGarbageDateThrowsException() {
        String garbage = "invalid-date";

        Exception exception = assertThrows(IllegalArgumentException.class, () -> {
            DateUtil.parse(garbage);
        });

        assertTrue(exception.getMessage().contains("Invalid date format"), "Should contain 'Invalid date format'");
    }

    //Null string should throw exception with specific message
    @Test
    void testParseNullDateThrowsException() {
        Exception exception = assertThrows(IllegalArgumentException.class, () -> {
            DateUtil.parse(null);
        });

        assertEquals("Input date string cannot be null", exception.getMessage());
    }

    //Null LocalDate in format() should throw exception
    @Test
    void testFormatNullDateThrowsException() {
        Exception exception = assertThrows(IllegalArgumentException.class, () -> {
            DateUtil.format(null);
        });

        assertEquals("Input date cannot be null", exception.getMessage());
    }

    //Valid LocalDate should return correct formatted string
    @Test
    void testFormatValidDate() {
        LocalDate date = LocalDate.of(2025, 7, 28);
        String expected = "28-07-2025";

        String result = DateUtil.format(date);
        assertEquals(expected, result, "Formatted string should match expected pattern");
    }
}
```

7. **Create a method to group bills by category and return a Map<String, List<Bill>>. Create getABillById().**

```java
    // Group a group of bills by category
    public static Map<String, List<Bill>> groupBillsByCategory(List<Bill> bills) {
    return bills.stream()
            .collect(Collectors.groupingBy(Bill::getCategory));
}

    // Retrieves a bill based on bill id
    public static Bill getABillById(List<Bill> bills, int billId) {
    for (Bill bill : bills) {
        if (bill.getBillId() == billId) {
            return bill;
        }
    }
    return null;
}
```

Report HACKATHON 2: 1st August, 2025

1. **TASK**

   Create a derived column late_fee in a view, calculated as 10% of amount if is_paid = false and due_date.

   Query:

```
CREATE OR REPLACE VIEW bills_with_late_fee AS
SELECT
   b.*,
   CASE
      WHEN b.is_paid = 0 AND b.due_date < SYSDATE
      THEN ROUND(b.amount * 0.10, 2)
      ELSE 0.00
   END AS late_fee
FROM bills b;
```

2. **TASK**

   Create bills table with bill_id(String, primary key) , bill_name (String), bill_category(String), due_date (date), amount(float), reminder_frequency (String), attachment (String), notes (String), is_recurring (boolean), is_paid (boolean), overdue_days (int), user_id(String, Foreign key)

   - user_id is a foreign key.
   - Insert 5 bills: include 2 categories (electricity, rent) and future/past due dates.
   - Add a unique constraint so that a user cannot have more than one bill with the same due_date & category.

   Query:

```
CREATE TABLE Bills (
   bill_id VARCHAR2(50) PRIMARY KEY,
   bill_name VARCHAR2(100),
   bill_category VARCHAR2(50),
   due_date DATE,
   amount FLOAT,
   reminder_frequency VARCHAR2(50),
   attachment VARCHAR2(255),
   notes VARCHAR2(255),
   is_recurring NUMBER(1) CHECK (is_recurring IN (0, 1)),
   is_paid NUMBER(1) CHECK (is_paid IN (0, 1)),
   overdue_days NUMBER(3),
   user_id VARCHAR2(50),
   CONSTRAINT uq_user_due_cat UNIQUE (user_id, due_date, bill_category)
);
```

```
INSERT INTO Bills VALUES (
   'B001', 'Ichigo Kurosaki', 'rent', TO_DATE('2025-07-28', 'YYYY-MM-DD'), 12000,
   'monthly', NULL, 'Paid in advance', 0, 1, 0, 'U001'
);

INSERT INTO Bills VALUES (
   'B002', 'Tanjiro', 'rent', TO_DATE('2025-08-05', 'YYYY-MM-DD'), 12000,
   'monthly', NULL, 'To be paid soon', 1, 0, 0, 'U001'
);

INSERT INTO Bills VALUES (
   'B003', 'Levi', 'electricity', TO_DATE('2025-07-30', 'YYYY-MM-DD'), 2500,
   'monthly', NULL, 'Late payment expected', 1, 0, 2, 'U002'
);

INSERT INTO Bills VALUES (
   'B004', 'Artyom', 'electricity', TO_DATE('2025-08-02', 'YYYY-MM-DD'), 2700,
   'monthly', NULL, 'Regular bill', 1, 0, 0, 'U003'
);

INSERT INTO Bills VALUES (
   'B005', 'Eren', 'rent', TO_DATE('2025-08-03', 'YYYY-MM-DD'), 12000,
   'monthly', NULL, 'Test entry', 0, 0, 1, 'U003'
);
```

3. TASK

Write a query to get all unpaid bills due within 5 days.

Query:

```
SELECT * FROM Bills
WHERE is_paid = 0 AND due_date BETWEEN TRUNC(SYSDATE) AND TRUNC(SYSDATE) +
5;
```

4. TASK

Write a query to join users and bills and display user name, bill category, and due date.

Query:

```
Select
   u.name,
   b.bill_category,
   b.due_date
from users u join Bills b
on u.user_id=b.user_id
```

HACKATHON 2 OUTPUT

```
Table BILLS created.


1 row inserted.


1 row inserted.


1 row inserted.


1 row inserted.


1 row inserted.
```

	BILL_ID	BILL_NAME	BILL_CATEGORY	DUE_DATE	AMOUNT	REMINDER_FREQUENCY	ATTACHMENT	NOTES	IS_RECURRING	IS_PAID	OVERDUE_DAYS	USER_ID
1	B002	Tanjiro	rent	05/08/25	12000	monthly	(null)	To be paid soon	1	0	0	U001
2	B004	Artyom	electricity	02/08/25	2700	monthly	(null)	Regular bill	1	0	0	U003
3	B005	Eren	rent	03/08/25	12000	monthly	(null)	Test entry	0	0	1	U003

```
View BILLS_WITH_LATE_FEE created.
```

```
SQL> select
  2      u.name,
  3      b.bill_category,
  4      b.due_date
  5  from users u join Bills b
  6  on u.user_id=b.user_id;

NAME
--------------------------------------------------------------------------------
BILL_CATEGORY                                      DUE_DATE
-------------------------------------------------- ---------
Sanjay Kumar
rent                                               28-JUL-25

Sanjay Kumar
rent                                               05-AUG-25

Priya Sharma
electricity                                        30-JUL-25


NAME
--------------------------------------------------------------------------------
BILL_CATEGORY                                      DUE_DATE
-------------------------------------------------- ---------
Amit Singh
electricity                                        02-AUG-25

Amit Singh
rent                                               03-AUG-25
```

(Contribution in Analytics.sql)

Task 1: Write a query to count users with more than 2 unpaid bills.

```
SELECT
   b.user_id,
   COUNT(*) AS unpaid_bill_count
FROM
   Bills b
WHERE
   b.is_paid = 0
GROUP BY
   b.user_id
HAVING
   COUNT(*) > 2;
```

Task 2: Design a bill_category table and normalize category column from bills using FK.

```
-- 1: Create BillCategory table
CREATE TABLE BillCategory (
   category_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
   category_name VARCHAR2(100) UNIQUE NOT NULL
);

-- 2: Populate BillCategory from distinct values in Bills.bill_category
INSERT INTO BillCategory (category_name)
SELECT DISTINCT bill_category FROM Bills;

-- 3: Add category_id column to Bills
ALTER TABLE Bills ADD (category_id NUMBER);

-- 4: Update category_id in Bills using lookup from BillCategory
UPDATE Bills b
SET category_id = (
   SELECT bc.category_id
   FROM BillCategory bc
   WHERE bc.category_name = b.bill_category
);

-- 5: Drop the UNIQUE constraint that includes bill_category
-- Find constraint name dynamically if needed; assuming it is: uq_user_due_cat
ALTER TABLE Bills DROP CONSTRAINT uq_user_due_cat;

-- 6: Drop old bill_category column
ALTER TABLE Bills DROP COLUMN bill_category;
```

```
-- 7: Drop any existing FK on category_id if error was encountered previously
BEGIN
   EXECUTE IMMEDIATE 'ALTER TABLE Bills DROP CONSTRAINT fk_bill_category';
EXCEPTION
   WHEN OTHERS THEN
      IF SQLCODE != -2443 THEN -- ORA-02443: Cannot drop constraint - nonexistent
         RAISE;
      END IF;
END;
/

-- 8: Add the foreign key constraint for category_id
ALTER TABLE Bills
ADD CONSTRAINT fk_bill_category
FOREIGN KEY (category_id)
REFERENCES BillCategory(category_id);
```