Dart Programming Assignment

Q1: Employee Salary Calculator

Question: Create a Dart program to calculate weekly salaries of employees. If hours worked > 40, extra hours are paid at 1.5 times the hourly rate. Use a class with constructor, methods, and a list of employees to display their final salaries.

Code:

```
class Employee {

  String name;
  double hourlyRate;
  double hoursWorked;


  Employee(this.name, this.hourlyRate, this.hoursWorked);


  double calculateSalary() {
    const double regularHours = 40.0;
    const double overtimeMultiplier = 1.5;
    double totalSalary;

    if (hoursWorked <= regularHours) {

      totalSalary = hoursWorked * hourlyRate;
    } else {

      double regularPay = regularHours * hourlyRate;
      double overtimeHours = hoursWorked - regularHours;
      double overtimePay = overtimeHours * (hourlyRate * overtimeMultiplier);
      totalSalary = regularPay + overtimePay;
    }

    return totalSalary;
  }
}

void main() {
```

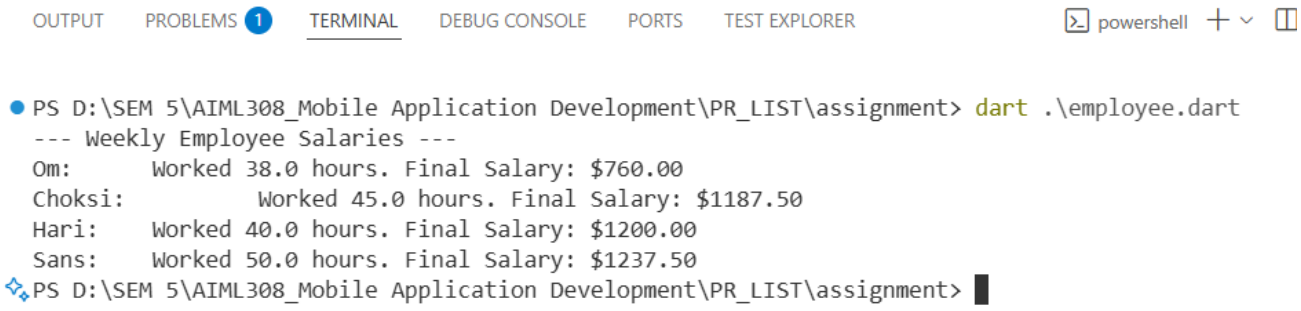
```
  List<Employee> employees = [
    Employee('Om', 20.0, 38.0),
    Employee('Choksi', 25.0, 45.0),
    Employee('Hari', 30.0, 40.0),
    Employee('Sans', 22.5, 50.0),
  ];

  print("--- Weekly Employee Salaries ---");

  for (var employee in employees) {
    double finalSalary = employee.calculateSalary();

    print(
        '${employee.name}: \t Worked ${employee.hoursWorked} hours. Final Salary:
\$${finalSalary.toStringAsFixed(2)}');
  }
}
```

OUTPUT:



Q2: Online Shopping Cart

Question: Design a shopping cart using a Map where items and prices are stored. Write a function to calculate total bill and handle invalid input like negative quantities or missing items using exceptions. Print the final total after handling errors.

Code:

```dart
import 'dart:io';

void main() {
  Map<String, double> products = {
    'Laptop': 78000.0,
    'Smartphone': 34000.0,
    'Headphones': 4500.0,
  };

  double totalAmount = 0.0;

  try {
    print('Enter the quantity for Laptop:');
    int laptopQty = int.parse(stdin.readLineSync()!);
    print('Enter the quantity for Smartphone:');
    int smartphoneQty = int.parse(stdin.readLineSync()!);
    print('Enter the quantity for Headphones:');
    int headphonesQty = int.parse(stdin.readLineSync()!);

    if (laptopQty < 0 || smartphoneQty < 0 || headphonesQty < 0) {
      throw FormatException('You cannot enter negative quantities!');
    }

    totalAmount += products['Laptop']! * laptopQty;
    totalAmount += products['Smartphone']! * smartphoneQty;
    totalAmount += products['Headphones']! * headphonesQty;

    print('Total cost: ₹${totalAmount}');
  } catch (e) {
    print('Error: $e');
  }
}
```

OUTPUT:

```
PS D:\SEM 5\AIML308_Mobile Application Development\PR_LIST\assignment> dart .\q2_shopping_cart.dart
Enter the quantity for Laptop:
2
Enter the quantity for Smartphone:
1
Enter the quantity for Headphones:
1
Total cost: ₹194500.0
PS D:\SEM 5\AIML308_Mobile Application Development\PR_LIST\assignment>
```

Q3: Student Performance Analyzer

Question: Store marks of students in a list of lists. Use higher-order functions (map, reduce, where) to calculate average marks, find highest and lowest scorer, and filter students above a threshold. Display all results neatly.

Code:

```dart
void main() {
  List<List<int>> studentScores = [
    [85, 90, 78],
    [75, 88, 92],
    [95, 100, 98],
  ];

  var averageScores = studentScores
      .map((scores) => scores.reduce((a, b) => a + b) / scores.length)
      .toList();
  var highestScore = studentScores
      .map((scores) => scores.reduce((a, b) => a > b ? a : b))
      .reduce((a, b) => a > b ? a : b);
  var lowestScore = studentScores
      .map((scores) => scores.reduce((a, b) => a < b ? a : b))
      .reduce((a, b) => a < b ? a : b);

  var topStudents = studentScores
      .where((scores) => scores.reduce((a, b) => a + b) / scores.length > 80)
      .toList();

  print('Average scores per student: $averageScores');
```

```
  print('Highest score in all subjects: $highestScore');
  print('Lowest score in all subjects: $lowestScore');
  print('Top students (average scores above 80): $topStudents');
}
```

OUTPUT:

Q4: Library Management System

Question: Create an abstract class Book with title and author, and an abstract method displayInfo(). Inherit it into EBook (with file size, format) and PrintedBook (with pages). Demonstrate polymorphism by calling displayInfo() for both types.

Code:

```
abstract class Book {
  String title;
  String author;

  Book(this.title, this.author);

  void displayInfo();
}

class EBook extends Book {
  double fileSizeMB;
  String fileFormat;

  EBook(String title, String author, this.fileSizeMB, this.fileFormat)
    : super(title, author);

  @override
  void displayInfo() {
    print(
      'EBook: "$title" by $author, Format: $fileFormat, Size: $fileSizeMB MB',
    );
```

```
  }
}

class PrintedBook extends Book {
  int pageCount;
  PrintedBook(String title, String author, this.pageCount)
    : super(title, author);
  @override
  void displayInfo() {
    print('Printed Book: "$title" by $author, Pages: $pageCount');
  }
}

void main() {
  Book ebook = EBook('Flutter for Beginners', 'Om', 5.5, 'PDF');
  Book printedBook = PrintedBook('Mastering Dart', 'Hari', 450);

  ebook.displayInfo();
  printedBook.displayInfo();
}
```

OUTPUT:

Q5: Railway Ticket Booking

Question: Simulate a ticket booking system using async programming. Use Future.delayed to check seat availability and confirm booking only if seats exist. Handle timeout and print "Booking Confirmed" or "Booking Failed" accordingly.

Code:

```
import 'dart:async';

int availableSeats = 3;


Future<bool> checkSeatAvailability(int seatsRequested) {
  print("\nChecking seat availability...");

  return Future.delayed(const Duration(seconds: 2), () {
    if (availableSeats >= seatsRequested) {
      print("Seats are available.");
      return true;
    } else {
      print("Sorry, not enough seats are available.");
      return false;
    }
  });
}


Future<String> confirmBooking(int seatsRequested) {
  print("Confirming your booking...");

  return Future.delayed(const Duration(seconds: 1), () {
    availableSeats -= seatsRequested;
    return "Booking Confirmed";
  });
}

Future<void> bookTickets(String user, int seatsToBook) async {
  print("--- ${user} started booking ${seatsToBook} seat(s) ---");

  try {
    String result = await Future(() async {
      bool areSeatsAvailable = await checkSeatAvailability(seatsToBook);

      if (areSeatsAvailable) {
        return await confirmBooking(seatsToBook);
```

```
      } else {
        throw Exception("Booking Failed: Not enough seats.");
      }
    }).timeout(const Duration(seconds: 4));

    print(" SUCCESS: $result for ${user}.");

  } on TimeoutException {
    print("FAILED: The booking process for ${user} timed out.");
  } catch (e) {
    print(" FAILED: ${e.toString().replaceFirst('Exception: ', '')}");
  } finally {
    print("--- ${user}'s booking attempt finished ---\n");
  }
}

void main() async {
  await bookTickets("Om", 2);

  await bookTickets("Hari", 2);

  await bookTickets("Sans", 1);
}
```

OUTPUT:

Q6: Bank Transaction System

Question: Build a BankAccount class with a private balance variable. Use getters and setters to safely deposit/withdraw money (no negative or overdraft allowed). Include null safety by keeping account holder name optional. Show operations with balance updates

Code:

```dart
class BankAccount {
  double _balance = 0.0;
  String? accountHolder;

  BankAccount({this.accountHolder});

  double get balance => _balance;

  void deposit(double amount) {
    if (amount > 0) {
      _balance += amount;
      print('Deposited: ₹${amount}, New Balance: ₹$_balance');
    } else {
      print('Deposit amount must be positive');
    }
  }

  void withdraw(double amount) {
    if (amount <= _balance && amount > 0) {
      _balance -= amount;
      print('Withdrew: ₹${amount}, New Balance: ₹$_balance');
    } else {
      print('Insufficient balance or invalid amount');
    }
  }
}
void main() {
  var account = BankAccount(accountHolder: 'Om');
  account.deposit(150000.0);
  account.withdraw(105000.0);
}
```

OUTPUT: