AI ASSISTED CODING:

LAB TEST:3

Set E2

Q1:
Scenario: In the Finance sector, a company faces a challenge related to code refactoring.
Task: Use AI-assisted tools to solve a problem involving code refactoring in this context.
Deliverables: Submit the source code, explanation of AI assistance used, and sample output.

Code:

```
# loan_calculator_refactored.py


import logging

from dataclasses import dataclass


logging.basicConfig(level=logging.INFO, format="%(asctime)s - %(levelname)s - %(message)s")


@dataclass

class CustomerProfile:

    income: float
    expenses: float
    credit_score: int
    existing_loans: int = 0

class LoanCalculator:
    def __init__(self, config=None):
        # Allow configuration to be customized externally
        self.config = config or {
            "high_credit_multiplier": 10,
            "medium_credit_multiplier": 5,
            "low_credit_multiplier": 2,
            "existing_loan_penalty": 1000,
            "min_loan_amount": 5000,
        }

    def calculate(self, customer: CustomerProfile) -> float:
        logging.info(f"Calculating loan for credit score:
{customer.credit_score}")

        disposable_income =
self._calculate_disposable_income(customer)
        multiplier = self._get_multiplier(customer.credit_score)
        loan_amount = disposable_income * multiplier
        loan_amount -= customer.existing_loans * self.config["existing_loan_penalty"]

        return max(loan_amount, 0) if loan_amount >= self.config["min_loan_amount"] else 0

    def _calculate_disposable_income(self, customer: CustomerProfile) -> float:
        if customer.income < customer.expenses:
            logging.warning("Customer expenses exceed income. No loan granted.")
            return 0
        return customer.income - customer.expenses

    def _get_multiplier(self, credit_score: int) -> int:
        if credit_score > 750:
            return self.config["high_credit_multiplier"]
        elif credit_score > 650:
            return self.config["medium_credit_multiplier"]
        else:
            return self.config["low_credit_multiplier"]
```

```
if __name__ == "__main__":
    profile = CustomerProfile(income=7000, expenses=3000,
credit_score=720, existing_loans=1)
    calculator = LoanCalculator()
    result = calculator.calculate(profile)
    print(f"Recommended Loan Amount: ${result:,.2f}")
```

output:

```
Recommended Loan Amount: $19,000.00
```

Explanation:

The code calculates interest for different account types (Savings, Fixed Deposit, Recurring Deposit) using **OOP concepts**.
An abstract class Account defines common variables (principal, rate, time) and an abstract method calculate_interest().
Each subclass implements its own interest formula.
A **factory function** get_account() creates the correct account object based on the type.
This refactoring removes long if-else statements, improves **readability**, **reusability**, and **maintenance**.

Q2:

Scenario: In the Hospitality sector, a company faces a challenge related to web frontend development.
Task: Use AI-assisted tools to solve a problem involving web frontend development in this context.
Deliverables: Submit the source code, explanation of AI assistance used, and sample output.

Code :

```
import React, { useState, useMemo } from "react";

export default function HotelBookingUI() {
 const HOTELS = [
  { id: 1, name: "Seaside Resort", location: "Goa", pricePerNight: 4500, rating: 4.6 },
  { id: 2, name: "City Center Hotel", location: "Mumbai", pricePerNight: 6500, rating: 4.2 },
  { id: 3, name: "Hill View Cottages", location: "Ooty", pricePerNight: 3200, rating: 4.8 },
 ];

 const [query, setQuery] = useState("");
 const [selectedHotel, setSelectedHotel] = useState(null);
 const [nights, setNights] = useState(1);
 const [bookings, setBookings] = useState([]);

 // Filtered search results
 const results = useMemo(() => {
  return HOTELS.filter((h) =>
   h.name.toLowerCase().includes(query.toLowerCase())
  );
 }, [query]);

 function confirmBooking() {
  const total = selectedHotel.pricePerNight * nights;
  const newBooking = {
   id: Date.now(),
   hotel: selectedHotel,
   nights,
   total,
   date: new Date().toLocaleString(),
  };
  setBookings((b) => [newBooking, ...b]);
  setSelectedHotel(null);
 }

 return (
  <div className="max-w-4xl mx-auto p-4">
   <h1 className="text-2xl font-bold mb-4"> Hotel Booking Demo</h1>

   {/* Search Bar */}
   <input
    value={query}
    onChange={(e) => setQuery(e.target.value)}
    placeholder="Search hotels..."
    className="w-full p-2 border rounded mb-4"
   />

   {/* Hotel List */}
   <div className="grid grid-cols-1 md:grid-cols-3 gap-4">
    {results.map((h) => (
     <div key={h.id} className="border p-4 rounded shadow">
      <h2 className="font-semibold text-lg">{h.name}</h2>
      <p className="text-sm text-gray-500">{h.location}</p>
      <p className="mt-1 text-sm">{h.rating}</p>
      <p className="mt-1 font-medium">₹{h.pricePerNight}/night</p>
      <button
       onClick={() => {
        setSelectedHotel(h);
        setNights(1);
       }}
       className="mt-3 px-3 py-1 bg-blue-600 text-white rounded"
      >
```

```
        Book
      </button>
    </div>
  ))}
</div>

{/* Booking Modal */}
{selectedHotel && (
  <div className="fixed inset-0 bg-black/40 flex items-center justify-center p-4">
    <div className="bg-white rounded-lg max-w-md w-full p-6">
      <h3 className="text-lg font-semibold">
        Booking — {selectedHotel.name}
      </h3>
      <p className="text-sm text-gray-600">
        Location: {selectedHotel.location}
      </p>

      <div className="mt-4">
        <label className="block text-sm">Nights</label>
        <input
          type="number"
          min={1}
          value={nights}
          onChange={(e) => setNights(Number(e.target.value))}
          className="mt-1 p-2 border rounded w-24"
        />
      </div>

      <div className="mt-4 flex justify-between items-center">
        <div>
          <div className="text-sm text-gray-500">Price / night</div>
          <div className="font-semibold">
            ₹{selectedHotel.pricePerNight}
          </div>
        </div>
        <div>
          <div className="text-sm text-gray-500">Total</div>
          <div className="font-semibold">
            ₹{selectedHotel.pricePerNight * nights}
          </div>
        </div>
      </div>

      <div className="mt-6 flex justify-end gap-2">
        <button
          onClick={() => setSelectedHotel(null)}
          className="px-3 py-1 border rounded"
        >
          Cancel
        </button>
        <button
          onClick={confirmBooking}
          className="px-3 py-1 bg-green-600 text-white rounded"
        >
          Confirm
        </button>
      </div>
    </div>
  </div>
)}

{/* Bookings List */}
```

```
      <section className="mt-8">
        <h3 className="text-lg font-medium">Recent Bookings</h3>
        {bookings.length === 0 ? (
          <p className="text-sm text-gray-500">No bookings yet.</p>
        ) : (
          <ul className="mt-3 space-y-2">
            {bookings.map((b) => (
              <li
                key={b.id}
                className="p-3 border rounded flex justify-between items-center"
              >
                <div>
                  <div className="font-semibold">
                    {b.hotel.name} — {b.nights} night(s)
                  </div>
                  <div className="text-sm text-gray-500">{b.date}</div>
                </div>
                <div className="font-semibold">₹{b.total}</div>
              </li>
            ))}
          </ul>
        )}
      </section>

      <footer className="mt-8 text-xs text-gray-500 text-center">
        Built for Hospitality Demo • Responsive UI
      </footer>
    </div>
  );
}
```
Output:

Booking — Seaside Resort

Location: Goa

Nights: [2]

Price/night: ₹4500

Total: ₹9000

[Cancel] [Confirm]

Explanation:

- The React app creates a **Hotel Booking UI** for the hospitality sector.
- Users can **search hotels**, **book rooms**, and **view recent bookings**.
- The modal allows entering the number of nights and shows **total price** dynamically.
- State management (useState, useMemo) handles search, selection, and bookings.
- The UI is built with **Tailwind CSS** for a clean, responsive layout.
- **AI tools** (like ChatGPT/Copilot) assisted in designing the component structure, state logic, and UI styling efficiently.

Sample Output:
Displays hotel list → booking modal → confirms booking → shows in "Recent Bookings" section.