|  |
| --- |
|  |
|  |
| **Project Based Learning Scenario** |
| **Assignment 1** |
| **Development Scenario 1: Personal Finance Tracker  Day 1: Introduction and Setup and Variables and Control Structures Task 1: Install Kotlin and configure IntelliJ IDEA. Verify the setup by running a "Hello, World!" program. Task 2: Explore Kotlin REPL (Read-Eval-Print Loop) to familiarize with Kotlin syntax and basic operations. Task 3: Create a Transaction class with properties such as amount, date, and category. Task 4: Implement control structures to categorize transactions (e.g., Food, Utilities, Entertainment) using when statements.  Day 2: Functions and OOP Basics Task 5: Write functions to add, delete, and edit transactions in a TransactionList class. Task 6: Develop a simple User class with methods to login and display a summary of expenses. Task 7: Use lambdas and higher-order functions to filter and sort transactions by date or amount. Task 8: Implement inheritance by creating specific transaction classes like Income and Expense that inherit from Transaction.  Day 3: Interfaces, Encapsulation, and Advanced Concepts / Collections and Generics Task 9: Define an Exportable interface with a method to export transaction data to CSV. Task 10: Apply encapsulation to Transaction properties using getters and setters ensuring sensitive data is protected. Task 11: Create generic functions to handle different types of collections (List, Set, Map) of transactions. Task 12: Utilize Kotlin's collection libraries to manage a collection of User objects, enabling the addition and removal of users.  Day 4: Null Safety and Exception Handling / Advanced Features (Extensions and Coroutines) Task 1: Implement null safety features to handle the absence of transaction data. Task 2: Write custom exception classes to handle errors related to transaction processing. Task 3: Create extension functions for the List<Transaction> class to calculate total expenses and incomes. Task 4: Use coroutines to handle simultaneous processing of importing and exporting transaction data without blocking the main thread.  Submission Guidelines: 1. Ensure that each answer is clear, concise, and reflects an understanding of the core concepts. 2. Diagrams can be hand-drawn and scanned or created using any digital drawing tool. 3. Provide references for any external sources used. Submit your work in a single PDF document by end of Module. 4. You must submit your code on gitlab by the end of next day**  **Development Scenario 2: Event Management System**  **Day 1: Introduction and Setup**  **Task 1: Set up the Kotlin development environment and write a simple Kotlin script to validate the setup.**  **Task 2: Experiment with Kotlin's string templates to create dynamic welcome messages.**  **Task 3: Define data types to represent event details such as name, date, and attendee count.**  **Task 4: Implement a basic user input flow to create new events using if and when statements.**  **Day 2: Functions and OOP Basics**  **Task 5: Design a EventManager class with methods to add and remove events.**  **Task 6: Create a Display interface with a method to show event details and implement it in the EventManager.**  **Task 7: Utilize higher-order functions to implement a simple notification system for event updates.**  **Task 8: Construct subclass SpecialEvent with additional features like VIP lists and premium services.**  **Day 3: Interfaces, Encapsulation, and Advanced Concepts / Collections and Generics**  **Task 9: Develop a Schedule class that uses interfaces to ensure that all event types can be scheduled and rescheduled.**  **Task 10: Secure the event data with proper encapsulation and visibility modifiers.**  **Task 11: Manage a collection of events allowing filtering by date or type using Kotlin's powerful collection operations.**  **Task 12: Use generics to create a flexible DataManager class capable of handling different data types, including attendees and events.**  **Day 4: Null Safety and Exception Handling/Advanced Features (Extensions and Coroutines)**  **Task 1: Ensure that the system gracefully handles null references when retrieving event data.**  **Task 2: Implement try-catch blocks to handle parsing errors when reading event dates and times.**  **Task 1: Write extension functions for the Event class to add features like tagging and categorization.**  **Task 2: Introduce coroutines to concurrently handle event bookings and cancellations.**  **Submission Guidelines:**  **1. Ensure that each answer is clear, concise, and reflects an understanding of the core concepts.**  **2. Diagrams can be hand-drawn and scanned or created using any digital drawing tool.**  **3. Provide references for any external sources used.**  **Submit your work in a single PDF document by end of Module.**  **4. You must submit your code on gitlab by the end of next day** |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| |  | | --- | |  | |  | | **Project Based Learning Scenario** | | **Assignment 1** | | **Development Scenario 1: Personal Finance Tracker  Day 1: Introduction and Setup and Variables and Control Structures Task 1: Install Kotlin and configure IntelliJ IDEA. Verify the setup by running a "Hello, World!" program. Task 2: Explore Kotlin REPL (Read-Eval-Print Loop) to familiarize with Kotlin syntax and basic operations. Task 3: Create a Transaction class with properties such as amount, date, and category. Task 4: Implement control structures to categorize transactions (e.g., Food, Utilities, Entertainment) using when statements.  Day 2: Functions and OOP Basics Task 5: Write functions to add, delete, and edit transactions in a TransactionList class. Task 6: Develop a simple User class with methods to login and display a summary of expenses. Task 7: Use lambdas and higher-order functions to filter and sort transactions by date or amount. Task 8: Implement inheritance by creating specific transaction classes like Income and Expense that inherit from Transaction.  Day 3: Interfaces, Encapsulation, and Advanced Concepts / Collections and Generics Task 9: Define an Exportable interface with a method to export transaction data to CSV. Task 10: Apply encapsulation to Transaction properties using getters and setters ensuring sensitive data is protected. Task 11: Create generic functions to handle different types of collections (List, Set, Map) of transactions. Task 12: Utilize Kotlin's collection libraries to manage a collection of User objects, enabling the addition and removal of users.  Day 4: Null Safety and Exception Handling / Advanced Features (Extensions and Coroutines) Task 1: Implement null safety features to handle the absence of transaction data. Task 2: Write custom exception classes to handle errors related to transaction processing. Task 3: Create extension functions for the List<Transaction> class to calculate total expenses and incomes. Task 4: Use coroutines to handle simultaneous processing of importing and exporting transaction data without blocking the main thread.  Submission Guidelines: 1. Ensure that each answer is clear, concise, and reflects an understanding of the core concepts. 2. Diagrams can be hand-drawn and scanned or created using any digital drawing tool. 3. Provide references for any external sources used. Submit your work in a single PDF document by end of Module. 4. You must submit your code on gitlab by the end of next day** | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |