Mobile Apps 2

Assignment Nine

Submission Date 23/12/2024

Percent 10%

Include all workings and references

Section A.

1. What are the key differences between Kotlin and Java Exception Handling and how does this affect the developer's code? (2.5 Marks)

Kotlin vs Java Exception Handling

- **Checked vs Unchecked Exceptions**:
    - **Java**: In Java, exceptions are categorized into *checked* and *unchecked* exceptions. Checked exceptions must be explicitly handled by the developer using a try-catch block or declared using throws in method signatures. On the other hand, unchecked exceptions (like RuntimeException) do not need to be declared or caught.
    - **Kotlin**: Kotlin does not have checked exceptions. All exceptions in Kotlin are unchecked, which means developers are not forced to declare them or handle them explicitly. This simplifies the code by reducing boilerplate associated with exception handling.
- **Syntax Differences**:
    - **Java**: Java uses the traditional try-catch-finally block for exception handling. Developers need to define the exception type in the catch block (e.g., catch (IOException e)).
    - **Kotlin**: Kotlin uses a similar try-catch block but with a more concise and flexible syntax. Developers don't need to explicitly declare exceptions in method signatures. Kotlin supports catching multiple exceptions in one catch block as well as using finally to release resources, much like Java.

- **Functionality and Impact**:
    - **Java**: In Java, the presence of checked exceptions can lead to more rigid code, as methods must either handle or declare every checked exception. This can increase the complexity of the code, especially in large systems where multiple checked exceptions are involved.
    - **Kotlin**: Kotlin's lack of checked exceptions allows developers to write cleaner and more flexible code, as they do not need to deal with exception declarations. However, it puts the responsibility on the developer to properly handle exceptions during runtime. This can lead to a potential issue where errors might be overlooked.

Impact on Developer's Code:

- **Java**: Developers must ensure that they handle or declare checked exceptions, which can lead to more verbose and complex code.
- **Kotlin**: Developers have more freedom as they are not required to declare checked exceptions, leading to cleaner and simpler code. However, this might also cause developers to neglect exception handling, which could lead to runtime errors.

2. Propose a means that would

    a. Grab all the details of a commercial form

    To grab all the details of a commercial form, we can propose a simple UI where a user can input necessary form details. These details can include:

    - **Product Name**
    - **Product Category**
    - **Price Range** (Minimum and Maximum Price)
    - **Delivery Time**
    - **Product Line**
    - **Express Delivery Option**

    The form will gather user input and submit the information for processing.

    b. Look into the relevant database of products based on several (but not all) criteria, such as min and max price (sorted), product line and express delivery time service.

    The next step would be to query a product database based on the criteria specified by the user. We would need to perform a search query that takes into account the following filters:

    - **Min and Max Price**: The database should filter the products that fall within the specified price range.
    - **Product Line**: Filter products based on a specified product line (e.g., electronics, clothing, etc.).
    - **Express Delivery**: Products that can be delivered within a specific time frame should be prioritized.

A SQL query example to retrieve such products might look like this:

```sql
SELECT *
FROM products
WHERE price BETWEEN minPrice AND maxPrice
AND productLine = productLine
AND expressDelivery = expressDelivery
ORDER BY price ASC;
```

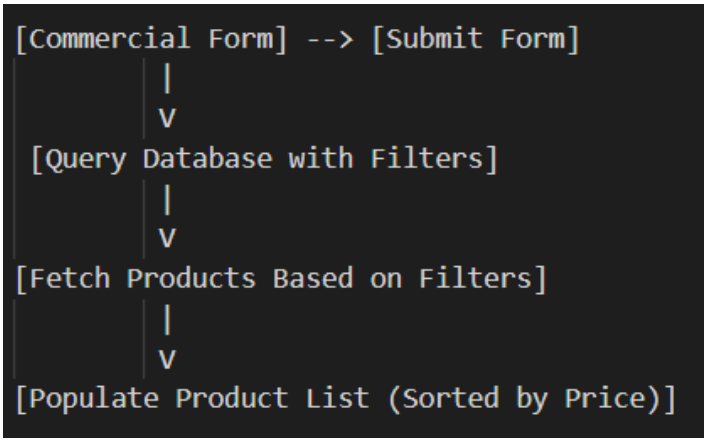
c. Populate the next page with a list of such products (sorted in ascending order of price by default).

Diagrams will assist in the answer (2.5 Marks)

Once the relevant products have been fetched from the database, we can populate the next page by displaying the product list in a UI element, such as a RecyclerView in Android.

- **Sorting by Price**: By default, the products will be sorted in ascending order based on their price. This ensures that users first see the cheapest products.
- **Displaying Product Details**: Each product in the list can display key information like name, price, delivery time, and availability of express delivery.

Diagram:
Here's a simple diagram illustrating the process flow:



Section B.

Use simple code in both answers

1. Show the use of Precondition Functions, with handling of the relevant exception by
    a. Catching it
    b. Throwing it (2.5 *marks*)
2. Create a custom exception that will be activated if a date value greater than 6 months from the present is selected / entered (2.5 *marks*)

The project code of Section B is on GitHub.

Repository: **ExceptionHandling**

References

1. Exception Handling in Android
2. Jetpack Compose - Official Documentation
3. Catch and handle exceptions
4. CompositeDateValidator
5. CalendarConstraints.DateValidator
6. CreateCredentialException
7. Kotlin Official Documentation: Exception Handling
8. Java Official Documentation: Exception Handling
9. SQL Basics: Introduction to SQL Queries
10. Android RecyclerView Documentation
11. RecyclerView in Android with Example