Android Assignment  
Set 1  
Try to solve any Three

**Q1.** Design and implement a Least Recently Used (LRU) Cache. A cache has a fixed capacity, and when it exceeds that capacity, it must evict the least recently used item to make space for the new one.

Implement the following operations:

* get(key): Return the value of the key if it exists in the cache, otherwise return -1.
* put(key, value): Update or insert the value. If the cache is full, remove the least recently used item before inserting.

Function Signatures:

class LRUCache {

public:

LRUCache(int capacity);

int get(int key);

void put(int key, int value);

};

Constraints:

* 1 <= capacity <= 3000
* 0 <= key, value <= 10^4
* Maximum number of operations: 10^5
* All operations must be done in O(1) time complexity.

Example:

Input:

LRUCache lru(2);

lru.put(1, 1);

lru.put(2, 2);

lru.get(1);

lru.put(3, 3);

lru.get(2);

lru.put(4, 4);

lru.get(1);

lru.get(3);

lru.get(4);

**Q2.** Problem Statement:

You are required to implement a simplified version of a HashMap (also known as an unordered map or dictionary), without using built-in hash table libraries like unordered\_map, map, dict, or similar.

Design a data structure that supports the following operations in average-case O(1) time:

* put(key, value) → Insert or update the value by key.
* get(key) → Return the value associated with the key. If not found, return -1.
* remove(key) → Remove the key from the map.

Function Signatures:

class MyHashMap {

public:

MyHashMap();

void put(int key, int value);

int get(int key);

void remove(int key);

};

Constraints:

* All keys and values are integers.
* 0 <= key, value <= 10^6
* Keys are unique within the map.
* Maximum operations: 10^5\
* Do not use built-in hash maps or dictionaries.

Example:

Input:

MyHashMap obj;

obj.put(1, 10);

obj.put(2, 20);

obj.get(1);

obj.get(3);

obj.put(2, 30);

obj.get(2);

obj.remove(2);

obj.get(2);

**Q3.** "You’ve been hired as a mobile developer for a startup building a Book Review App. Your task is to implement a minimum viable product (MVP) version of the app that allows users to browse, view details, and save books locally for offline access."

Requirements & Features:

1. Architecture (must use Java):

* Use either MVVM or Clean Architecture.
* Separation of layers: UI, domain, data.
* Use ViewModel, Repository, UseCase (if using Clean Architecture).

2. Core Features:

* Book List Screen
  + Fetch list of books from a fake API (you can provide JSON file or a mock endpoint).
  + Show title, author, and thumbnail.
* Book Detail Screen
  + Show full description, rating, and image.
* Save to Favorites
  + User can "favorite" a book.
  + Saved books are stored using Room (SQLite).
* Offline Mode
  + Bookmarked books can be viewed offline.

3. Tech Stack & Constraints:

* Java only (no Kotlin).
* Use Retrofit for networking (or manual JSON parsing to test parsing skills).
* Room for persistence.
* LiveData or Observables for reactive UI.
* No external libraries for image loading (simulate loading via placeholders).

**Q4.** You are tasked with creating a mini solar system visualization using OpenGL (ES 2.0+ or 3.0) that demonstrates your understanding of the graphics pipeline, transformations, and shaders.

Requirements:

1. Render a simple solar system scene:
   * A central Sun that remains static at the center.
   * At least two planets orbiting the Sun at different speeds and distances.
   * One of the planets must have a moon orbiting it.
2. Implement custom shaders:
   * Write your own vertex and fragment shaders using GLSL.
   * The Sun should use a shader-based glow or pulsing effect.
   * Planets and moon can have textures or simple gradient coloring via shaders.
3. Apply transformations:
   * Use matrix transformations to handle orbiting and rotation animations.
   * Planets must rotate on their axis while orbiting the Sun.
4. User interaction:
   * Implement camera controls:
     + Rotate the view with mouse drag or touch input.
     + Optional: Zoom in/out with mouse scroll or pinch.
5. Performance:
   * The application should run smoothly (~30 FPS or higher).
   * Use VBOs/VAOs or equivalent for rendering efficiency.

**Submission Guidelines**

Create a GitHub repository

Include all source code

Provide a comprehensive README

Submit repository link

Deadline:- 3 days