**Assignment 1**

**CSC241: Object Oriented Programing**

Assigned Date: 03-10-2025

**Due Date: 10-10-2025**

Maximum Marks: 10

CLO:SO(1):Demonstrate fundamental principles and concepts of object-oriented programming.

**Description**

In this lab you will design and implement, from scratch and using only Java arrays, a city-wide cinema management system that practices encapsulation, enums, array indexing, and multi-class collaboration. Your model has four core entities, Seat, Screen, Cinema, and CityCinema, plus two demo drivers. At the foundation, a Seat represents a single chair identified by a human-friendly ID in the form row-col such as 3-007. Seat must use a type-safe enumeration for classification (REGULAR, PREMIUM, VIP, RECLINER), store a price, track availability, and provide safe state changes through methods that book and cancel a booking. Its toString should present a short summary with ID, type, price, and availability.

A Screen is a physical auditorium backed by a jagged two-dimensional Seat array. A simple default is five rows whose lengths increase as 10, 11, 12, 13, 14. When a Screen is constructed it must materialize the entire grid, assign seat types by a clear policy (front rows REGULAR, middle rows PREMIUM, penultimate rows VIP, last row RECLINER), and derive prices from type so the system demonstrates both structure and behavior without manual per-seat setup. Use a pricing policy that is visible in output and easy to verify: REGULAR 500 PKR, PREMIUM 750 PKR, VIP 1000 PKR, RECLINER 1200 PKR. Screen must locate a seat by ID or by coordinates, book and cancel seats, count total and available seats overall and by type, and provide two displays: a compact layout view that shows each seat with an availability marker and a detailed listing that prints every seat using toString. Whenever a method needs to return multiple seats, return a plain Seat[] built with the classic two-pass technique: first count matches, then allocate an array of that size and fill it.

A Cinema aggregates several Screen instances under a branch name using a plain Screen[] plus a logical size counter. Because collections are not used, grow capacity manually by allocating a larger array and copying elements when needed. Support construction with a fixed number of default Screens to showcase array-based preloading, plus operations to add a named Screen with custom row lengths, find a Screen by name or by index, forward booking and cancellation to a particular screen and seat ID, compute totals and availability overall and by SeatType, and print clear reports that iterate through each screen’s layout or detailed list. The Cinema toString should compress its state into a single readable line that includes name, number of screens, total seats, and available seats.

CityCinema coordinates many Cinema branches within a named city using the same arrays-only approach: maintain a Cinema[] with a logical size, grow it manually, and provide methods to add and remove cinemas, find a cinema by name, relay booking and cancellation requests down to the correct cinema and screen, and compute city-wide aggregates. Include a discovery helper that scans the city for the first available VIP seat and returns a human-friendly description such as Cinema-2 > Screen-1 > Seat 3-007 (VIP, 1000.00 PKR). If you need to return multiple results, expose them as arrays rather than collections. Provide a city preload method that builds a fixed array of cinemas, each with a fixed array of screens, each screen materializing a two-dimensional seat array, so the program never starts empty.

Your demonstration must feel like a real Pakistani deployment from the first line of main. In ScreenDemo or CinemaDemo, begin by building CityCinema objects for Karachi, Lahore, and Islamabad, all preloaded strictly through arrays. Use realistic example names to make the output readable, such as Atrium Saddar and Nueplex DHA in Karachi, CineStar Township and CineStar Gulberg in Lahore, and Centaurus Megaplex in Islamabad. Print compact layouts for each city, then simulate a typical workflow: book seat 3-007 in Karachi at Cinema-1 on Screen-1, attempt to book the same seat again to show correct rejection, cancel the booking, and print the affected layouts again to confirm the state change. Add a small search task that prints the first available VIP seat in Lahore. Separately, write SeatDemo that focuses only on Seat: construct a few seats of different types, verify that new seats are available by default, perform bookings and cancellations, adjust a price where appropriate, and print results with toString so that the enum, price, and availability logic is obvious.

A key deliverable is data that is visible at startup. Each program must load already created data using arrays: for each city, create a fixed array of Cinema objects; each Cinema preloads a fixed array of Screen objects using jagged row arrays; each Screen materializes a two-dimensional Seat array and assigns types and prices according to the stated policy (REGULAR 500 PKR, PREMIUM 750 PKR, VIP 1000 PKR, RECLINER 1200 PKR). First, create the starting data using arrays (cities, cinemas, screens, seats). After that, add code for actions like booking and canceling seats. By the end of Lab 5, your program should handle several Pakistani cities. Each city should have multiple cinemas, each cinema should have multiple screens, and each screen should have seats of different types and prices. The program must let you book and cancel by seat ID, show how many seats are available and their prices at each level, and print clear layouts and summaries. Do all of this using only arrays (no Java Collections).

You must submit the following classes: Seat, Screen, Cinema, CityCinema, SeatDemo, and ScreenDemo or CinemaDemo.

Submission Guidelines:

Submit your work through the Google Form at <https://forms.gle/6EhLcMZ8UYKBf5Ty9> . In the form, provide your **Name** and **Registration Number**, and upload **one ZIP file** that contains your entire submission.

Your ZIP file should be named clearly, such as RegistrationNumber\_Lab5\_Cinema.zip. Inside this ZIP, include your Java source code and a Word document. Keep the package structure consistent so the code compiles and runs without changes.

* **Source code (folder structure):**  
  **Word document (Lab5\_Report.docx):**  
  This file must contain:
  + **Title page** with the assignment title “Lab 5 — Cinema Management System”, the date (October 1, 2025), your Name, Registration Number, Section, Course, Instructor, and Department or University.
  + **Java class listings** or short code excerpts that demonstrate your implementation choices.
  + **Sample run output** that shows the program starting with preloaded data, printing layouts for the cities, booking a seat by ID, attempting a double booking, canceling the booking, and showing the updated state. Screenshots or copied console text are both acceptable.

Use a Java-safe package by converting to lowercase and replacing hyphens with underscores: **sp25\_bcs\_xxx**.  
Example: registration **SP25-BCS-123** → package **sp25\_bcs\_123**, repo/folder name **SP25-BCS-123\_Lab5\_Cinema**.

Make sure your code compiles and runs from the src folder with the package **sp25\_bcs\_xxx**. A simple way to build and run is to compile all files into an out directory and then execute your main class:

* **Compile:**  
  javac -d out src/sp25\_bcs\_xxx/\*.java
* **Run:**  
  java -cp out sp25\_bcs\_xxx.ScreenDemo  
  or  
  java -cp out sp25\_bcs\_xxx.CinemaDemo

Replace xxx with your three digits. Ensure every .java file starts with package sp25\_bcs\_xxx; and resides in src/sp25\_bcs\_xxx/.

Before you upload, confirm that:

* The ZIP contains the full src directory and the Word document.
* The Word document includes the title page and the sample run output.
* Class names and package names match what you used in your code.
* The program runs without additional configuration.