KLE Society's

KLE Technological University



**Open Ended (OE) Assessment Report**

**On**

**STOCK PORTFOLIO MANAGEMENT SYSTEM**

**Object Oriented Programming (20ECSC204)  
Object Oriented Programming Lab (20ECSP203)**

Submitted by

| **Name** | **Roll no** | **SRN** |
| --- | --- | --- |
| Chandni kumari | 11 | 02FE22BCS026 |
| Prashant Uppar | 21 | 02FE22BCS069 |
| Wilfred Antonio Borges | 69 | 02FE22BCS177 |
| Yukta Sannaki | 71 | 02FE22BCS179 |
| **Team Number**: A11 | | |

Faculty In-charge:

**Prof. Vaishali Y. Parab**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Academic year 2022-23

| **1.** | **Introduction** | |
| --- | --- | --- |
|  | 1.1 | Overview of the problem statement  Managing a stock portfolio involves tracking various financial instruments such as equities and bonds. Investors need a systematic and user-friendly solution to add, store, and display these instruments efficiently. The goal is to create a portfolio management system that encapsulates these functionalities while ensuring memory safety and extensibility. |
|  | 1.2 | Features of Application  ### Features of the Portfolio Management Application  1. Portfolio Management Interface:  The application features a console-based interface that allows users to manage a portfolio containing stocks and bonds. This interface is designed to be straightforward and user-friendly, enabling users to interact with the portfolio through text-based commands. Users can add new financial instruments to their portfolio by selecting the appropriate options from a menu. Additionally, the interface provides a way to view the contents of the portfolio, displaying detailed information about each stock and bond held. This makes it easy for users to monitor and manage their investments, keeping track of their financial instruments in a centralized and accessible manner.  2. Different Types of Financial Instruments:  The application supports the management of two main types of financial instruments: Equities (stocks) and Bonds. These instruments are modeled using a polymorphic approach, where a base class named `Stock` provides common functionality, and derived classes `Equity` and `Bond` implement specific behaviors and attributes for each instrument type. This design leverages the principles of object-oriented programming, allowing the application to treat different types of financial instruments uniformly while still catering to their unique characteristics. Polymorphism ensures that the system can easily be extended to include additional types of financial instruments in the future.  3. Smart Pointers for Memory Management:  The `Portfolio` class utilizes `shared\_ptr` to manage stocks and bonds, ensuring that memory management is handled automatically. Smart pointers, specifically `shared\_ptr`, are used to keep track of the ownership of dynamically allocated objects, ensuring that they are properly deallocated when no longer needed. This approach prevents memory leaks, which can occur when objects are not correctly deallocated, leading to inefficient memory usage and potential application crashes. By using smart pointers, the application ensures robust and efficient memory management, contributing to the overall stability and reliability of the system.  4. Template Function for Adding Stocks:  The function template `createAndAddStock` simplifies the process of creating and adding various types of financial instruments to the portfolio. This function uses variadic templates and perfect forwarding to accept a wide range of constructor arguments, making it highly flexible and adaptable to different needs. By leveraging C++ template features, the application can handle the instantiation of different financial instruments dynamically, reducing the need for repetitive code and enhancing maintainability. This template function demonstrates the power of modern C++ features in creating efficient and reusable code components.  5. Exception Handling:  The application includes a custom exception class `InvalidStockException` to manage invalid operations, such as incorrect stock type inputs. Exception handling is crucial for building robust applications, as it allows the system to gracefully handle errors and provide meaningful feedback to the user. By defining custom exceptions, the application can catch specific error conditions and respond appropriately, such as prompting the user to correct their input or displaying an error message. This improves the overall user experience by ensuring that the application behaves predictably and reliably even in the face of invalid inputs or other exceptional situations.  6. Friend Function for Displaying Portfolio:  The friend function `showPortfolioDetails` is used to display the details of the portfolio, illustrating the use of friend functions to access private members of a class. In C++, friend functions are a way to grant specific functions access to the private and protected members of a class. This approach is useful for functions that need to interact closely with the internal state of a class without exposing these details publicly. `showPortfolioDetails` enables the application to present comprehensive portfolio information to the user while maintaining encapsulation and adhering to object-oriented design principles.  7. User Interaction and Input Validation:  The application prompts users to enter details for the financial instruments they wish to add to their portfolio, such as the name and price for stocks or the issuer and interest rate for bonds. It includes basic input validation to ensure that the entered data is appropriate and meaningful. For instance, the application might check that numerical inputs are within a valid range or that textual inputs are non-empty. Error messages are provided for invalid choices, guiding the user to correct their input. This focus on user interaction and input validation helps to prevent errors and ensure that the data entered into the system is accurate and useful, contributing to the overall reliability and usability of the application. |
| **2.** | **Design**  **2.1 Class Diagrams** | |
|  |
| **4** | **Implementation** | |
|  | 4.1 | Results |