**UNIVERSITY OF MALTA**
**FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY**
**DEPARTMENT OF COMPUTER SCIENCE**

**CPS2004 Object Oriented Programming**

# Assignment 2023/24 - September submission

---

**Instructions:**

1. This is an **individual assignment**.
2. You may be required to demonstrate and present your work to an exam board.
3. A soft-copy of the report in pdf format must be uploaded to the VLE upload area, along with the code repository, archived in a single `.zip` file by the deadline given below. A `README.md` file must be included in the root directory of the repo describing the content's organization.
4. Reports (and code) that are difficult to follow due to low quality in the writing-style/organisation/presentation will be penalised.
5. Assignment queries are to be made strictly on the assignment's forum on VLE.
6. This assignment comprises 100% of the final CPS2004 assessment mark.
7. You are to allocate 50 to 60 hours for this assignment.
8. The report must be organized according to the task sequence, and for each, clearly presenting the required UML diagrams and a high-level overview identifying which classes implement the required functionality and the corresponding source files. The full source code files for each task must also be privately shared on gitlab with mmarrkv set as a developer collaborator.
9. A video with a maximum duration of five minutes should be created using screen recorder software and uploaded to your Google Drive account, and privately shared with mark.vella@um.edu.mt . A link to the video should be included in the first page of the report. The video should give an overview

of the code repositories, how to build the software artefacts for each of the tasks and demonstrating their functionality. Both voice-overs and textual annotations are accepted.

10. The firm submission deadline is **6th September 2024 at NOON**.

1. **A Financial Portfolio Application developed in C++ and Java.** (Total-40 marks)

   You are required to develop software for a Financial Portfolio Application to provide the following functionality:

   - Create/read/update/sell financial assets.
   - Create/read/update financial intermediaries.
   - Record daily snapshots of all the financial assets to the asset history.
   - Display historical listings per asset filtered by start and end dates.
   - Calculate the total value annual returns for the currently-held financial assets.
   - Save/restore the full application state.
   - A command-line interface (CLI) OR a test driver demonstrating the application's functionality comprehensively .

   At a minimum, your application design should feature the following classes:

   - A Financial Asset class hierarchy, with the common attributes kept inside an abstract base class (required attributes: name, value), and three concrete subclasses: Stock (required attributes: ticker symbol, shares quantity, dividend yield), Bond (required attribute: interest rate, maturity date) and Mutual Fund (required attributes: expense ratio). The Financial Asset class hierarchy includes the following polymorphic methods: `calculateAnnualReturn()` - an asset-specific method to calculate the asset's annual return (any reasonable formula is accepted for the sake of this assignment and no marks will be deducted for financially incorrect formulas), and `displayAsset()` - which produces a string representation of the asset's type and all its attributes.
   - A Financial Intermediary class hierarchy with common attributes kept inside an abstract base class and three concrete subclasses: Broker, Bank, and MutualFundManager, to be associated with Stock, Bond and Mutual Fund assets respectively. The Financial Intermediary class hierarchy includes the following polymorphic method: `displayIntermediary()` - which produces a string representation of an intermediary's details.

- A Portfolio class which aggregates the currently owned assets, their historical snapshots, and the associated intermediaries. This class also implements methods which retrieve the current assets and their historical snapshots, as well as calculating the total annual return for the entire asset portfolio.
- A CLI class that handles console interaction, OR ELSE a test driver class.

Tasks: -

(a) Design UML class diagrams capturing the application functionality and structure as described above. You need to flesh out the unspecified details in a reasonable manner, just enough to render the application demonstratable.

**[10 marks]**

(b) Implement the application in C++, with classes, hierarchies, associations and dependencies as identified in the UML diagrams.

**[15 marks]**

(c) Implement a Java version of the application.

**[15 marks]**

*The code deliverable for this task should not yet include the enhancements required in the subsequent tasks. This instruction applies for all tasks. You are advised to read the entire assignment specification prior to start working on it.*

The use of CMake and Maven to build the C++ and Java applications respectively is compulsory for this and all the subsequent tasks. The `CMakeLists.txt` and `pom.xml` files must be made available with the code repository. The C++ source files should be compiled using **C++20**, while the Java sources should be compiled using **Java 17**.

2. **Refactoring the Financial Portfolio Application.** (Total-25 marks)

Now that a functional first version of the implementation is available in two languages, you are required to refactor **one** of the codebases, thereby improving codebase maintainability and robustness.

Tasks: -

(a) Create a dedicated class hierarchy of immutable classes to represent the historical snapshots of the financial assets.

**[10 marks]**

(b) Create the corresponding immutable objects using the Builder Pattern.

**[10 marks]**

(c) Explain how the above measures improve codebase maintainability and robustness.

**[5 marks]**

3. **Portable data persistence.** (Total-15 marks)

   Enhance the refactored version with data persistence and interoperability with other applications by using protocol buffers-based serialization.

   Tasks: -

   (a) Create a protobuf file defining the data structures to be persisted.

   **[5 marks]**

   (b) Create a Facade exposing just the `save()` and `load()` methods to persist/restore the application state using protobuf serialization.

   **[5 marks]**

   (c) Implement an Asset Viewer Application *in the language not chosen for refactoring* (so if you earlier refactored the Java version, this application should be written in C++). This application should only provide functionality to view the currently owned financial assets (i.e. no history view is required).

   **[5 marks]**

4. **Distributed computation.** (Total-20 marks)

   Enhance the Financial Portfolio Application for distributed deployment, spreading the computational load among multiple nodes over gRPC. In this first attempt, partition the application to provide the facility to calculate the annual return of individual Stock assets on a remote node (but still tested on the same machine).

   Tasks: -

(a) Create a protobuf file defining the interface for the remote service.

**[5 marks]**

(b) Update the main application to use the remote service. Make use of a new class, StockSvc, to be used as a Facade for the generated service stub. The service should be consumed by the Portfolio class.

**[5 marks]**

(c) Implement the remote service in the *other language not used to implement the main application* (i.e. the language to implement the remote service matches the language used for the Asset Viewer Application in the previous task.)

**[10 marks]**

---



*end of assignment*