THE UNIVERSITY OF SYDNEY

# SOFT2412/COMP9412 Assignment

**Agile Software Development Practices**

## 1 Summary

**Weight:** 10%
**Open:** August 25 2025 at 11:59 PM
**Due:** September 08 2025 at 11:59 PM
**Late policy:** Work past due date has penalty of 5% of available marks per day (except with approved special consideration, special arrangements, or simple extension). No submissions will be accepted after September 15 11:59.

## 2 Context

This is a collaborative software development project where teams of 4 students will work together to build a currency converter application while demonstrating proficiency in Git/GitHub, Gradle, and JUnit testing.

## 3 Overview

Build a simple currency converter application working in teams of 4 students. You should follow the *Integration-Manager* workflow. The provided repository:

https://github.sydney.edu.au/SOFT2412S2/currency-converter

is the *blessed repository* in this workflow. Each group maintains a repository that corresponds to the *integration-manager* repository. You can designate one person as the integration manager, or can take turns managing it, but there should be only one such repository.

Each student maintains and submits their own public individual repository in Sydney Github that is forked from the integration manager. This should be readable by other team members. You can maintain your private repository in your local machine.

## 4 Objectives

- Demonstrate Git branching and merging skills
- Collaborate effectively using GitHub
- Build Java applications using Gradle
- Write and execute unit tests with JUnit
- Integrate code from multiple developers

# 5 Team Structure

Form teams of 4 students. *If your team size is smaller or larger than four, please talk to your tutor as to how to divide your tasks. Please note that teams that are different from four members are actively discouraged, and should be formed only on direction from the tutor.* Each team member will have to implement one function each in the following Java classes:

- Core conversion logic (`CurrencyConverter.java`)
- User interface (`UserInterface.java`)
- Rate display (`RateDisplay.java`)
- Data validation (`DataValidator.java`)

In order to make the application work, all the group members should jointly work on:

- Main application (`App.java`)

Each member who implements a function **must also develop the respective unit test/s** in `src/test`.

# 6 Application Requirements

Build a console-based currency converter with the following features. The application should run in the terminal/command line (no GUI required) and be implemented as a single program using Java.

## 6.1 Core Functionality

1. Currency Conversion
   - Support conversion between 4 currencies: USD, EUR, GBP, AUD
   - Use hardcoded exchange rates (e.g., USD to EUR = 0.85) — you may define your own fixed rates, but they must be consistent throughout the program
   - Ensure conversion calculations are accurate (e.g., correct decimal places and rounding)
2. User Interface
   - Display a main menu with the following options:
     (a) Convert Currency
     (b) View Exchange Rates
     (c) Exit
   - After each operation, return to the main menu until the user chooses Exit
   - Clearly display results and messages so the user knows what to do at each step
3. Input Validation
   - Validate currency codes (only accept: USD, EUR, GBP, AUD)
   - Validate amounts (positive numeric values only, decimals allowed)
   - Handle invalid inputs gracefully (e.g., show an error message and prompt the user again without crashing)
4. Exchange Rate Display
   - Show current exchange rates in a formatted table
   - Display rates for all supported currency pairs (including reverse and same-currency conversions)
   - Show the exchange rate for a specific currency pair
   - Display all exchange rates as a list for supported currency pairs
   - Show 1.00 for same-currency conversions

## 6.2 Example Output

```
=== Currency Converter ===
1. Convert Currency
2. View Exchange Rates
3. Exit
Enter choice: 1

Enter amount: 100
From currency (USD/EUR/GBP/AUD): USD
To currency (USD/EUR/GBP/AUD): EUR
Result: 100.00 USD = 85.00 EUR

Returning to main menu... (-> Design the logic on your own)

=== Currency Converter ===
1. Convert Currency
2. View Exchange Rates
3. Exit
Enter choice: 2

Exchange Rates (base = 1 unit)
        USD     EUR     GBP     AUD
USD     1.00    0.85    0.75    1.30
EUR     1.18    1.00    0.88    1.53
GBP     1.33    1.14    1.00    1.73
AUD     0.77    0.65    0.58    1.00

Returning to main menu...
```

## 6.3 Additional Notes

- You *should not* fetch live rates; Please use hardcoded rates.
- Format numeric outputs to 2 decimal places.
- The program should continue running until the user selects the Exit option.
- Code should be readable, documented, and well-structured, following best practices taught in the course.

# 7 Technical Requirements

1. Individual Repository Management — Follow the Integration manager, **Main variant**, *not the Github variant*.
    - Each team forks their own blessed repository from the provided repository: https://github.sydney.edu.au/SOFT2412S2/currency-converter
    - Each student forks their own public repository from the group repository.
    - All team components must be integrated into each student's repository
    - Use Git branches for feature development
    - Merge branches using merge commits (not fast-forward merges)
2. Git Workflow Requirements — Follow the Integration manager, **Main variant**, *not the Github variant*.
    - Create feature branches with meaningful names (e.g., `feature/conversion-logic`)

- Make regular commits with clear commit messages
- Use merge commits to integrate features into main branch
- Repository history must show evidence of branching and merging

3. Gradle Build System
    - Project must build successfully using the gradle `build` command
    - Application must run using gradle `run` command
    - All dependencies should be managed through Gradle
4. JUnit Testing
    - Write unit tests for all major functionality
    - Tests must execute successfully using gradle `test` command
    - Aim for meaningful test coverage (recommended 60% at least)
    - Include both positive and negative test cases

# 8 Implementation Guidelines

## 8.1 Suggested Component Responsibilities

- `CurrencyConverter.java`
  - convert(double amount, String from, String to) - perform currency conversion
  - getCurrencies() - return array of supported currencies
  - getExchangeRate(String from, String to) - get rate between currencies
- `UserInterface.java`
  - start() - main application loop
  - showMenu() - display menu options
  - handleConversion() - guide user through conversion process
  - showExchangeRates() - display rates as per the user requirement
- `RateDisplay.java`
  - showAllRatesTable() - Display formatted table of exchange rates
  - showRate() - Display exchange rate for specific currency pair
  - showAllPairs() - Display all exchange rates for supported currency pairs
  - showSameCurrencyRates() - Display rates for same-currency conversions
- `DataValidator.java`
  - isValidCurrency(String currency) - validate currency codes
  - isValidAmount(String amount) - validate numeric amounts
  - parseAmount(String input) - safely parse user input
- `App.java`
  - main() method - application entry point
  - Initialize and coordinate other components
  - Handle overall application flow

# 9 Submission Requirements

Each student submits their own **GitHub repository under University of Sydney** **https://github.sydney.edu.au** containing:

1. Complete Working Application
    - All four components integrated and functional
    - Successful build with gradle `build`
    - Working application via gradle `run`
2. Comprehensive Test Suite
    - Unit tests for all major functionality
    - Tests pass with gradle `test`
    - Evidence of testing both success and error cases
3. Clear Git History
    - Multiple feature branches visible in history
    - Merge commits showing integration of features
    - Individual commit history demonstrating personal contribution
4. Updated Documentation (README.md). This should contain:
    - Team members
    - Explaining how to build, run, and test (Quick Start)
    - How your team divided the work
    - What you personally implemented
    - Name the merge commits you have done

# 10 Assessment Criteria – SOFT2412

## 10.1 Git Usage and Collaboration (60%)

- Commit comments (20%): Proper use of commit messages
- Branching Strategy (20%): Proper use of feature branches with meaningful names
- Merging and Integration (20%): Evidence of merge commits and successful code integration

## 10.2 Application Functionality (10%)

- Code Quality (10%): Clean, readable, well-organized code, with comments
- No marks if the code does not compile.

## 10.3 Testing (10%)

- Test Coverage : Comprehensive tests covering major functionality
- No marks if tests do not run any tests.

## 10.4 Build System (20%)

- Gradle Integration (10%): Successful build, run, and test execution

# 11 Assessment Criteria – COMP9412

## 11.1 Git Usage and Collaboration (50%)

- Commit comments (20%): Proper use of commit messages
- Branching Strategy (10%): Proper use of feature branches with meaningful names
- Merging and Integration (10%): Evidence of merge commits and successful code integration

## 11.2 Application Functionality (10%)

- Code Quality (10%): Clean, readable, well-organized code, with comments
- No marks if the code does not compile.

## 11.3 Testing (10%)

- Test Coverage : Comprehensive tests covering major functionality
- No marks if tests do not run any tests.

## 11.4 Build System (20%)

- Gradle Integration (10%): Successful build, run, and test execution

## 11.5 Reflection (10%)

- You are required to include a file `REFLECTION.md` in your Github repository that details the pros and cons of using this model of development vs simply committing directly into the main branch of the repository.

## General Advise

- **Review Tutorials and Lectures:** Begin by reviewing the tutorials and lectures. Remember that everything you need for each component has already been covered in this unit.
- **Understand the Fundamentals:** Go through the revision slides on Ed and make sure that you understand all of the content covered so far.
- **Ask Questions:** If you have any questions or uncertainties about the material covered, don't hesitate to ask on Ed for clarification and a TA will get back to you shortly.

## Academic Declaration

By submitting this assignment, you declare the following:
*I declare that I have read and understood the University of Sydney Student Plagiarism: Coursework Policy and Procedure, and except where specifically acknowledged, the work contained in this assignment/project is my own work, and has not been copied from other sources or been previously submitted for award or assessment.*
*I understand that failure to comply with the Student Plagiarism: Coursework Policy and Procedure can lead to severe penalties as outlined under Chapter 8 of the University of Sydney By-Law 1999 (as amended). These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgment from other sources, including published works, the Internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.*
*I realise that I may be asked to identify those portions of the work contributed by me and required to demonstrate my knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.*
*I acknowledge that the School of Computer Science, in assessing this assignment, may reproduce it entirely, may provide a copy to another member of faculty, and/or communicate a copy of this assignment to a plagiarism checking service or in-house computer program, and that a copy of the assignment may be maintained by the service or the School of Computer Science for the purpose of future plagiarism checking.*