MAJOR PROJECT

OVERVIEW

The SOFT3202 Major Project will comprise a custom piece of software you will develop during the semester, which you will further modify in the 'long release' window during the exam period with an extra set of instructions. You will be building a desktop GUI Java application, which consumes web APIs to achieve specified functionality.

The specification of your software is governed by a pair of APIs that vary from student to student. Each API has specific requirements within a wider framework that is the same for all students.

Submission

As done in previous assignments, you will use a private GitHub repository located at: https://github.sydney.edu.au. The repository must be named **SCD2_2022** and you must add the following collaborators:

- jbur2821
- aest9988
- agha0431
- efis3423
- hzen5475
- phao5814

Your major project submission should be in the directory <root>/major_project.

This submission may only be modified up until 23:59 on the Sunday of Week 13 (29MAY). At this time, your work on the given features must have been completed – your work during the 48 hours of the exam itself must be based on the state of your repository at this time to be accepted. Note there will be milestone assessments of your work at the end of Week 9, 11, and 13.

Specifications

There are 6 parts to the instructions you will be sent over the course of your work on this project.

- This document: explains the work you will be doing up to the end of Milestone 1. The same for all students.
- An email indicating what APIs you have been assigned to work with. You should receive this soon after this document is released.
- Your tutor feedback requirements after Milestone 1. Adds any extra requirements the tutor believes to be necessary after reviewing your initial submission.
- The Optional Features specification, also after Milestone 1. Indicates the optional features required for Credit/Distinction/High Distinction results.
- The 'exam period modification' specification. Explains the work you will be doing in the 48-hour exam window during the exam period. This will be released prior to the exam so you know what to expect, but will not tell you what the modification is. The same for all students.
- An email indicating what your personal modification is. This will be sent at the beginning of the exam window.

As always Ed announcements and API specific clarifications should also be considered part of the specification.

THE API

The APIs are divided up into 2 categories – an 'input' list API, and an 'output' list API (note that you may need to both send and receive information from either category, it is a thematic difference rather than a purely technical one). You will be assigned 1 API from each category - in essence your application will take information from one API and output it in some way through another.

You grade is initially determined by the completion level of the API. Each API will follow a similar scheme:

HURDLE

- You must implement the required features as indicated for your assigned input and output APIs.
- Your application must run without crashing.
- Your UI must be functional, with no raw JSON displayed, features are clearly visible.
- Your code must be free of obvious, major design flaws. Examples of these (which are not a complete list) would be:
    - 'God classes' or 'god methods' that know too much or do too much. Break your implementation down into components.
    - Spaghetti code, where functionality or data that should be kept together is split over many methods or classes
    - Major breaches in encapsulation – public attributes, liberal use of instanceof, etc

In most cases detailed UI requirements will not be mandated, but you should ensure that the presented information of your entity is easily readable, and the interaction is simple.

PASS:

- You must pass the Test Checkpoints at the 4 key submission dates:
    - Milestone 1
    - Milestone 2
    - Milestone 2 Resubmission
    - Exam
- To pass a Test Checkpoint your Model code must be fully tested (not using real database or HTTP calls) at the time of submission. If no submission is made or the submission was unsuccessful, you must still have a working test suite at the appropriate commits in your repository history:
    - 1 where you have the Pass criteria implemented but no further
    - 1 at your final in-semester commit
    - 1 at the end of the exam
- You must have correct Model-View separation in your design
- You must correct anything your tutor tells you to correct after Milestone 1. If you do not submit Milestone 1, you must see your tutor for these corrections as soon as you have a working submission. If you do not do this in time for Milestone 2, your maximum possible cap for Milestone 2 will be Pass. If you do not do this in time for the Milestone 2 resubmission, your maximum possible cap for the Exam will be Pass.

CREDIT:

- You must cache the appropriate data to a local SQLite file database
- This database must be created from scratch by your application when it is run with no database file present
- You must complete 2 standard optional features

DISTINCTION:

- You must use concurrency such that your UI remains responsive throughout slow network calls
- You must additionally complete 1 advanced optional feature

HIGH DISTINCTION:

- You must use a multi-layer architecture such as MVC, MVP, MVVM
- You must additionally complete 1 difficult optional feature

Other requirements

Note that the above requirements are ***feature requirements*** only, and should be considered as a 'ceiling' – that is, if you do not implement the Credit requirements, your maximum mark is 64%. How close to your earned ceiling your mark reaches, is based on the design, process, testing, style, and documentation requirements listed later, as well as from your work during the exam period itself. See the week 8 lecture for details.

Further information about the Credit and higher tiers will be available after Milestone 1.

INPUTS

https://www.abstractapi.com/holidays-api

Your GUI must begin with a world map (use of https://github.com/controlsfx/controlsfx World Map View is recommended). From this world map view the user must be able to select a country by clicking on it. All input API requests will use the chosen country (changing countries requires quitting and re-starting).

Your application GUI must then display a calendar. This calendar should display month-at-a-time, and have selection available from the user to change year and month, for any date from 1/1/1970-31/12/2037. Each day should display the date (1-31), and the user must be able to click on each day to determine whether there are any holidays on that day. If there are none, the day should indicate 'no holidays'. If there are, the day should be shaded a different colour, and the day should indicate the name(s) of the holiday(s). If the user clicks on the name of a holiday, a popup window should indicate a description of the holiday.

'long' form report output from this input is:

a list of all the *already* known holidays in a given month for the given country.

'short' form report output from this input is:

a list of each day that has an already known holiday in the given month for the given country.

What to cache: holidays for any given country+date combination.

https://coinmarketcap.com/api/documentation/v1/

Your application GUI must begin with an empty list. The user can 'add currency' at which point a separate window showing a list of all active coinmarketcap cryptocurrencies is displayed (Name and Symbol only). The user can select one of these cryptocurrencies from this window and close it, at which point the selected currency is added to the main window list. This main window list shows more detail about the cryptocurrency selected - logo (displayed visually), name, symbol, description, date launched, website (a url that can be clicked, launching the default browser - see HostServices in JavaFX).

Once multiple currencies have been selected, the user is now able to ask for conversions between them. The user must be able to input an amount of a given currency, select a target currency, and be told what the conversion rate is between those 2 currencies, and what the result from their given input value would be. The user can clear the list, or remove individual currencies from the list.

'long' form report output from this input is:

A report on a given conversion: including all main-window information about the 2 selected currencies, conversion rate, starting value, finishing value.

'short' form report output from this input is:

A report on a given conversion: including all selection-window information (name and symbol only) about the 2 selected currencies, conversion rate, starting value, finishing value.

What to cache: Info on each given currency. Not conversions.

https://currencyscoop.com/api-documentation

Your application GUI must begin with an empty list. The user can 'add currency' at which point a separate window showing a world map is displayed (use of https://github.com/controlsfx/controlsfx World Map View is recommended). Selecting a country in that map and closing the window adds the fiat currency for that country to the main window list. This main window list shows info on all selected currencies (currency code, name). Once multiple currencies have been selected, the user is now able to ask for conversions between them. The user must be able to input an amount of a given currency, select a target currency, and be told what the conversion rate is between those 2 currencies, and what the result from their given input value would be. The user can clear the list, or remove individual currencies from the list.

'long' form report output from this input is:

A report on a given conversion: including all information about the 2 selected currencies including countries, conversion rate, starting value, finishing value.

'short' form report output from this input is:

A report on a given conversion: including all main-window information (name and code only) about the 2 selected currencies, conversion rate, starting value, finishing value.

What to cache: All conversion rates

https://www.abstractapi.com/ip-geolocation-api

Your application GUI must display a world map (use of https://github.com/controlsfx/controlsfx World Map View is recommended), and an empty list. The user must be able to enter IP addresses, at which point a display is added to the main window indicating the IP address, city, region, and country. In the world map, the country is highlighted in red, and a dot is placed at the correct longitude/latitude. The user can keep checking IP addresses, which keep adding to the list, red countries, and location dots. The user can clear the list, or remove individual IP addresses from the list.

'long' form report output from this input is:

A list of all checked IP addresses with all displayed info (not the map)

'short' form report output from this input is:

Info on the last checked IP address with all displayed info (not the map)

What to cache: Results for each IP address

https://developer.oxforddictionaries.com/

Your application GUI should allow the user to enter a word in English. Upon entering a word, the application should retrieve the entry for that word. If the word does not have an entry, it should check for lemmas. If there is 1 lemma, it should retrieve the corresponding entry. If there are multiple lemmas, it should offer the choice to the user then retrieve the selected entry. If there are no lemmas the user should be told there are no matching words. Once an entry is retrieved, it should be displayed with all provided information neatly formatted. Any audio pronunciation files should be able to be played by the user. Any synonyms or antonyms should be able to be clicked by the user to retrieve that entry and display it.

The GUI should provide a 'breadcrumbs' feature that allows the user to see their selected word history, and to navigate backwards and forwards in this history. This display should visually distinguish between movement where the user clicked a synonym/antonym, and where the user searched for a new word.

'long' form report output from this input is:

Info on the currently selected entry, with all data for that word included.

'short' form report output from this input is:

A text representation of the user's entry history

What to cache: entry and lemma results

https://www.alphavantage.co/

Your application should present the user with a search bar with autocomplete functionality. This should not be based on each character typed, but rather upon pressing enter/clicking search a list of matching company symbols is displayed. The user can then either continue typing/searching or select one of the matching companies.

Once a company is selected, the application should display information about the selected company (symbol, name, description). Along with this the application should create and display line charts for the following cash flow values for the selected company: operatingCashflow, capitalExpenditures, profitLoss, dividendPayout, netIncome. These line charts should be displayed one at a time with the user being able to select any chart to display. The user can search for a new company at any time.

'long' form report output from this input is:

Info on the currently selected company, with all line chart data

'short' form report output from this input is:

Info on the currently selected company (symbol and name only), along with the most recent value of the currently selected line chart.

What to cache: Info for each selected company.

https://developer.edamam.com/food-database-api

The application should start by having the user enter an ingredient. The user should then be presented with a list of matching foods to choose from. Once the user has made a choice, the user should then be shown the nutritional values for that ingredient, along with a running total of each. The user can continue to search for and select ingredients to add, which are all visible in the application (this is not required to be all at once), with the running total for all currently selected ingredients.

The ingredients selected have both a 'measure' attribute and a value for that measure. These measures and values should be selectable by the user for each ingredient, and the nutritional information adjusted accordingly.

The user is able to indicate a maximum for any given nutritional value. The application should be able to show the user a % stacked bar chart comparing each nutritional value against its set maximum

'long' form report output from this input is:

> The list of currently selected ingredients (name, measure, value) along with the total nutritional value.

'short' form report output from this input is:

> The list of currently selected ingredients (name, measure, value)

What to cache: The nutritional information for specific food items for specific measures and values.

https://developer.marvel.com/

Your application should present the user with a search bar with autocomplete functionality. This should not be based on each character typed, but rather upon pressing enter/clicking search a list of characters that start with the given string is displayed. The user can then either continue typing/searching or select one of the matching characters.

Upon selecting a character, the application should display information about that character (thumbnail - displayed as an image, and name). A list of comics that character has appeared in (by comic name and volume) should also be displayed.

The user should be able to click a comic, and see what characters are also in that comic. They should then be able to select a character, and the character display should update with the new characters info (thumbnail and name). This character->comic->character navigation should continue as long as there are comics/characters to select.

The GUI should provide a 'breadcrumbs' feature that allows the user to see their selected character/comic history, and to navigate backwards and forwards in this history. Duplicates do not need to be handled in any special way; they can reappear in the list.

'long' form report output from this input is:

> Info on the currently selected character - name, list of comics, stories, events.

'short' form report output from this input is:

> A text representation of the user's character/comic search history

What to cache: Character and Comic query results

https://developers.pandascore.co/ - Overwatch

The application should allow the user to search for teams by name. If no team is found, the user should be told there is no match and allowed to search again. If a team is found, the user should be shown information about that team (name, image - displayed, not just the uri, location). For each player in the team also display (professional name, image - displayed).

The user should be able to select the team or any individual player, and list their upcoming matches (dates, number of games, opponent name) - up to 50.

Where the opponent is a team, the user should be able to select the team and list it in place of the original team. This team->match->team navigation should continue as long as there are matches/teams to select.

The GUI should provide a 'breadcrumbs' feature that allows the user to see their selected match/team history, and to navigate backwards and forwards in this history. Duplicates do not need to be handled in any special way; they can reappear in the list.

'long' form report output from this input is:

Info on the currently selected team - name, list of players, upcoming matches.

'short' form report output from this input is:

A text representation of the user's team/match search history

What to cache: Team and match query results

https://developer.riotgames.com/ use OC1 region

The application begins by allowing the user to search for a given summoner (player) by exact name - see https://www.leagueofgraphs.com/rankings/summoners/oce. Given a selected summoner, the application should display the name, summoner level, and for each queue type with data for that summoner, league wins, and league losses. Use https://ddragon.leagueoflegends.com/cdn/12.7.1/img/profileicon/<profileIconID>.png to obtain the profile id and display it also.

There should also be a set of pie charts showing the balance between wins and losses for each queue type with data - these should be displayed one at a time with the user being able to select any chart to display.

The user can search for a new summoner at any time.

'long' form report output from this input is:

The summoner name, level, id, [queue type, wins, losses, win%] for each queue type

'short' form report output from this input is:

The summoner name, level, and win% for the currently selected queue type

What to cache: Summoner and league query results.

https://open-platform.theguardian.com/

Your application should present the user with a search bar with autocomplete functionality. This should not be based on each character typed, but rather upon pressing enter/clicking search a list of matching tags (matching the starting string) is displayed. The user can then either continue typing/searching or select one of the matching tags.

With a tag selected, the user then searches for content within the currently selected tag.

The results of that content search are then displayed in a list, with Web Title and date.

The user is able to select any content entry, and the web Url is launched in the default browser (see HostServices in JavaFX).

The user is able to change tag/content search at any time. Tag searching clears content searching.

'long' form report output from this input is:

The currently selected tag along with the list of content.

'short' form report output from this input is:

The currently selected tag and the selected content entry only.

What to cache: Tag content search results (not partial tag name search results).

https://www.weatherbit.io/api

Use the list of cities >15,000 population from https://www.weatherbit.io/api/meta.

Your application should start by displaying a world map (use of https://github.com/controlsfx/controlsfx World Map View is recommended). The user can then enter a city name. This city name selection should be auto-completing, allowing the user to make a specific selection based on country + state + city.

Upon making a selection the current weather for that city is displayed (temperature, wind speed & direction, clouds, precipitation, air quality). The location of the city (longitude + latitude) should be marked on the world map with an icon from https://www.weatherbit.io/api/codes based on the reported weather.

The user can continue to check city weather. Each new city should be displayed on the map together, and all previously searched city weather data should be present in the GUI.

The user can clear searched data, which will clear the map, and also remove specific cities from the results.

'long' form report output from this input is:

Each searched city weather data.

'short' form report output from this input is:

The most recent searched city weather data.

What to cache: Weather data for a given city.

https://data.rijksmuseum.nl/object-metadata/api/

Your application should re-implement the advanced search at https://www.rijksmuseum.nl/en/search/advanced with all fields. This search should yield for the user a list (0-*) of collection items. The search result entries should include a small image (webImage), and the name of the entry. The user is then able to select each item, see the full size webImage (displayed, not just a uri), long title, date (yearEarly), plaque description in English, and principal maker(s) name(s).

The user is able to mark an entry as 'favourite', adding it to a list which can be viewed separately to search results. This favourites list can be added to across multiple searches, cleared, and individual entries removed.

'long' form report output from this input is:

The favourites list including all displayed data (not the images)

'short' form report output from this input is:

The currently selected entry from search or favourites (long title, date, principal makers' names)

What to cache: Selected entry data (not search results)

OUTPUTS

https://www.twilio.com/

Send a 'short' form report to a single configured phone number.

https://sendgrid.com/solutions/email-api/

Send a 'long' form report to a single configured email address.

https://pastebin.com/doc_api

Save a 'long' form report to pastebin.

https://apidocs.imgur.com/

Create a QR code embedding the text of a 'short' form report. You may choose to use https://github.com/zxing/zxing

Upload this QR code as an image

DUMMY API

Just as in Task 2 and for exactly the same reasons, you will need to implement a dummy version of both your input and your output APIs. These dummy APIs do not need to be dynamic, simply outputting reasonable looking data in order to demonstrate your work is sufficient. These should be independent – so live input – dummy output, vice versa, both live, or both dummy should work. These should be selectable on the command line with 2 arguments in sequence aka:

gradle run --args="offline online"

with the first controlling the input API, and the second controlling the output API.

API KEY ENVIRONMENT VARIABLES

For each API you require an API key – you should ensure your application looks for those keys in the following **environment variables**, as that is where your marker will be setting them:

- Basic API key for input API: INPUT_API_KEY
- Input API app id (if required): INPUT_API_APP_ID
- Imgur client id: IMGUR_API_KEY
- Imgur secret: IMGUR_API_SECRET
- Pastebin API key: PASTEBIN_API_KEY
- Sendgrid API key: SENDGRID_API_KEY
- Sendgrid target email: SENDGRID_API_EMAIL
- Twilio Token: TWILIO_API_KEY
- Twilio SID: TWILIO_API_SID
- Twilio send from number: TWILIO_API_FROM
- Twilio send to number: TWILIO_API_TO

If your API requires a token or configuration not listed above, ask on Ed. No manual editing of configuration files, INIs, entering tokens on application start etc should be required to run your software.

DESIGN REQUIREMENTS

You will be assessed on your application of the design principles taught throughout the semester. One aspect you should focus on is ensuring you have separation between your Model, and your View. This at a minimum will be necessary to achieve the testing/process requirements.

Your design will be assessed against the 3 goals this unit focuses on – maintainability, extensibility, and modularity (MEM), with intermediate principles such as SOLID, GRASP, and others applied to evaluate these.

Design Patterns

You are not explicitly required to use any design patterns – in contrast, you should NOT use design patterns where there is no benefit to the 3 MEM goals by doing so. If there is a net benefit however, you should of course use them.

TESTING NOTES

Do not overdo your testing. You have already demonstrated your ability to test for coverage in a black-box scenario in Task 1 and the Testing Assignment – you do not need to do this again. The previous stipulations that were present in those assignments, namely:

- test simple getters/setters
- test constructors
- exhaustively test defensive programming
- exhaustively mock any path

do not apply in this assessment. You should ensure you test for the most critical ‘good’ paths through your software. You should mock all external dependencies (both APIs, even the dummy versions (so you can call verify), and any database), but do not need to test independently otherwise, and your mocks only need to be sufficient for your current implementation to be tested (white box mocking). You should test defensive programming anywhere a human user or view is giving input, but nowhere else is required.

STYLE & COMMENTING

You should generally follow Google's Java Style guide - https://google.github.io/styleguide/javaguide.html

For marking purposes, the following sections and their sub-sections should be considered enforceable: 2, 5, 7. Outside of these, variation from the guide is acceptable, but must be a) readable, and b) consistent.

DOCUMENTATION

You must include a readme file in your repository. This can optionally be in readme.txt or readme.md formats. This readme file must:

- Include citations for any code you have used that you did not write
- Include citations for any code you DID write, but in another unit of study or prior attempt at this unit (note that you must also indicate where you have done either of the above with comments in the locations you have done so)
- A list of commit SHA hashes and URIs (not just SHA!) indicating your previously marked submissions, or testing checkpoints where these do not line up with actual submissions
- Any quirks to running your application the marker needs to know about
- An indication of which level of features you are claiming to have implemented (HURDLE, PASS, CREDIT, DISTINCTION, HIGH DISTINCTION)

ADDING DEPENDENCIES

As at the writing of this document the following libraries are allowed to be included in your build.gradle file:

- JUnit 5
- Hamcrest
- Mockito
- SQLIte-JDBC
- JavaFX
- Gson

There will be a pinned Ed post where you can make requests to add to this list – these will be reviewed and either accepted or rejected on a whole-class basis. Note that any 'sdk' style libraries that model a given web API you are using will not be accepted.

Including an unapproved dependency will result in your submission being rejected.

GETTING STARTED

As always, the first steps should be to explore your assigned APIs. You will require an API token for each of these. While they all have a free tier that should be more than sufficient for this assignment, there are request limits you should be aware of – do not waste this!

You should then spend some time on your overall design. While the exam work you will complete in the exam period itself is intended to only take a few hours, this will only be true if you set yourself up for success with an extensible design in this phase.

You should then work from the bottom up. Achieve the Pass requirements before adding in the Credit database caching, and only once you have that working should you implement the Distinction concurrency.

These requirements are stepped – if you do not meet the pass requirements it does not matter if you have a working database or threads.

Avoiding a Zero

Your application must compile and run. This is the fundamental requirement both of your application at the end of Week 11, and your application at the end of the exam window. If your application does not compile, or does not run, your project score will be zero. You are being implicitly assessed on your ability to make use of the fundamental tools and techniques of reliable software construction and design in order to avoid this.

The most obvious of these is your own manual testing – you should obviously be able to run your application yourself using the same gradle run command your marker will be using. Ensure you check this on multiple machines, and especially outside of any IDE execution environment.

Next is version control. You should work on your application in steps as indicated above. Note down the commits you have made where you are confident you have met certain requirements and your application runs successfully. This means that if you hit the end of week 13 and do not have one of the more advanced features finished, or break something tweaking the design, you can roll back those changes. During the exam period itself you may find that the more advanced features are making your modification more difficult – you are allowed to roll back changes and use an older version during the exam period as well if you wish.

Finally comes programmatic testing. You are not being recommended to write tests before you write code to meet some archaic teaching requirement, it is a widespread technique used in software development globally to ensure code not only works more reliably once it is done, but to ensure that it reaches the point of viability faster – this is part of the Agile development toolkit. By ensuring that you have tests running early, and that you keep running them as you develop, crippling errors are less likely to end up in what you deliver.