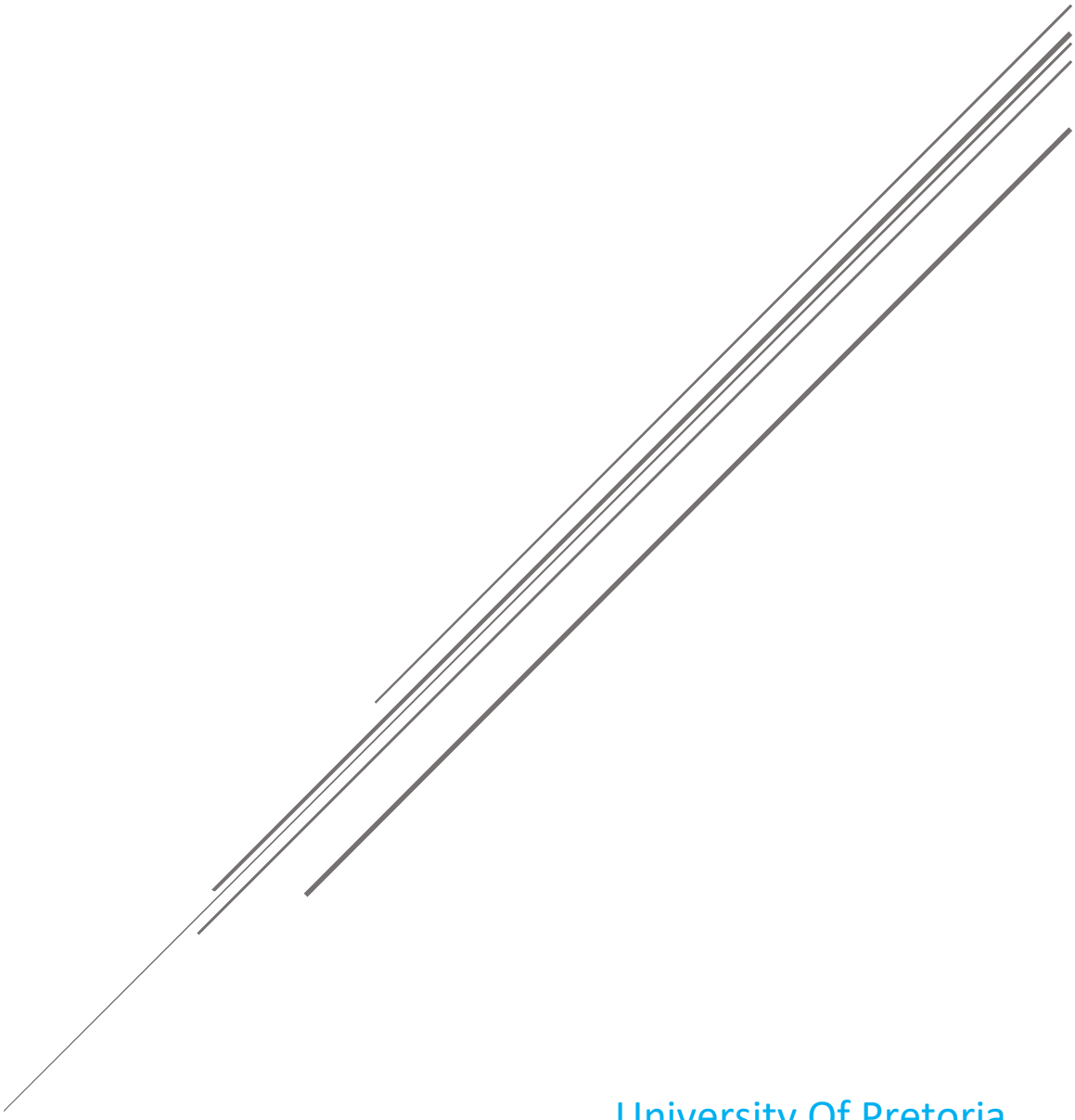
THE PATTERN PIRATES

Franco Dreyer (u22520016) | Marie Pretorius (u21565342) | Monika Theiss (u21434558) | Shannon Venter (u21451070) | Tristan Potgieter (u21437302)



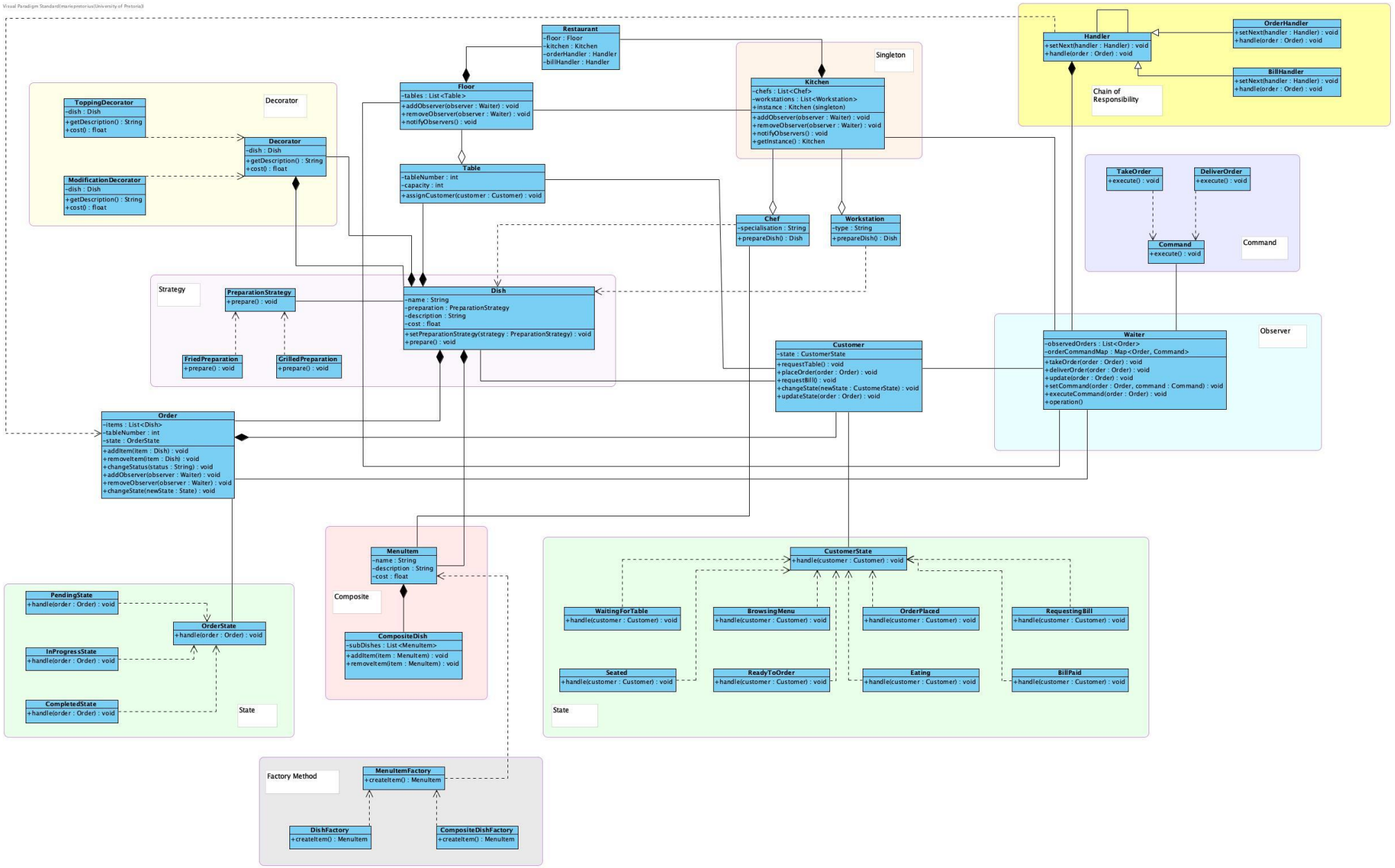
University Of Pretoria
COS214

WE ARE BUILD-A-PLATE. A RESTAURANT WHERE YOUR WILDEST DREAMS CAN BE REALIZED IN THE FORM OF A MEAL. YOU THINK IT, WE MAKE IT.

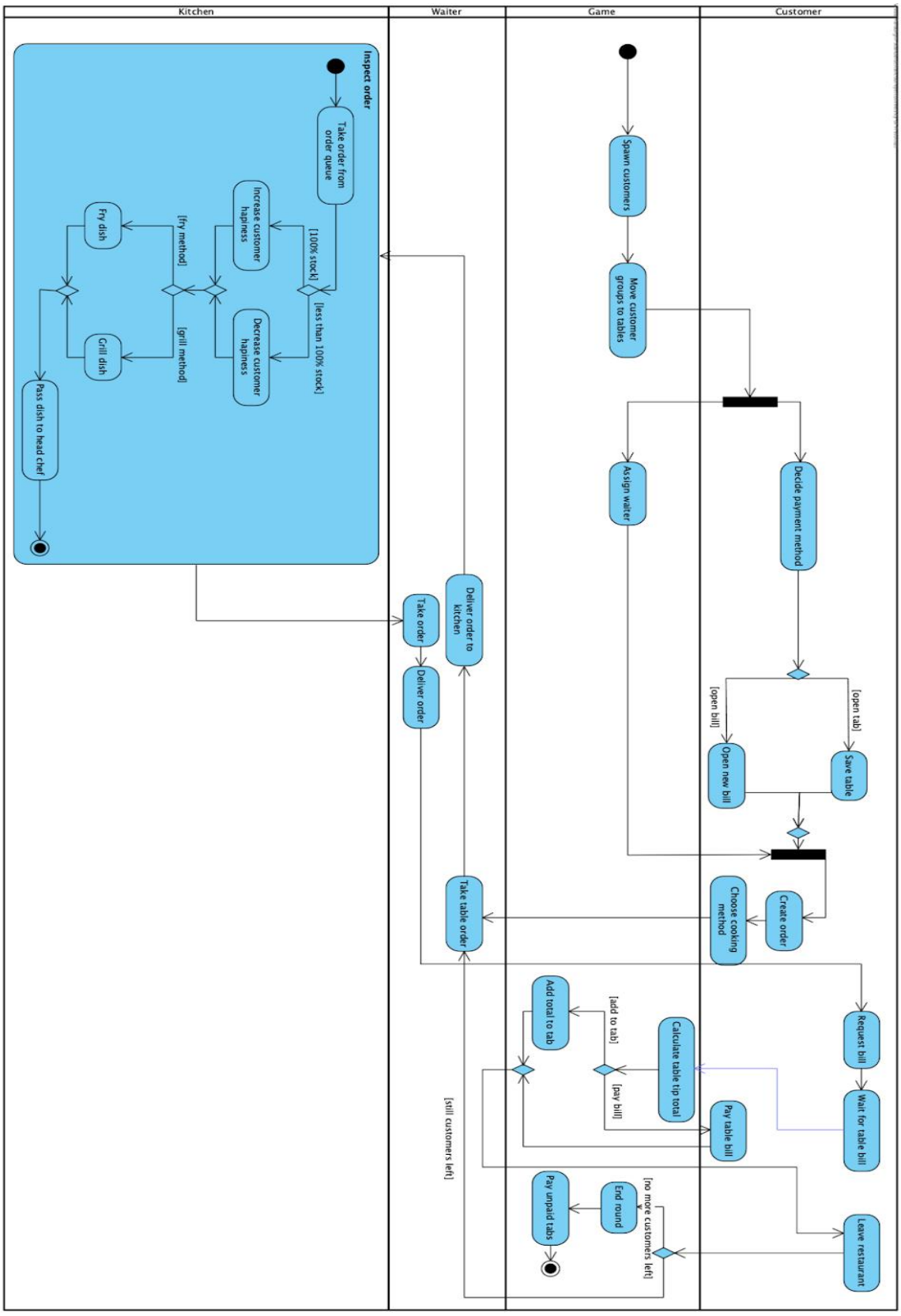
TASK 1:

The initial design of our system includes the following:

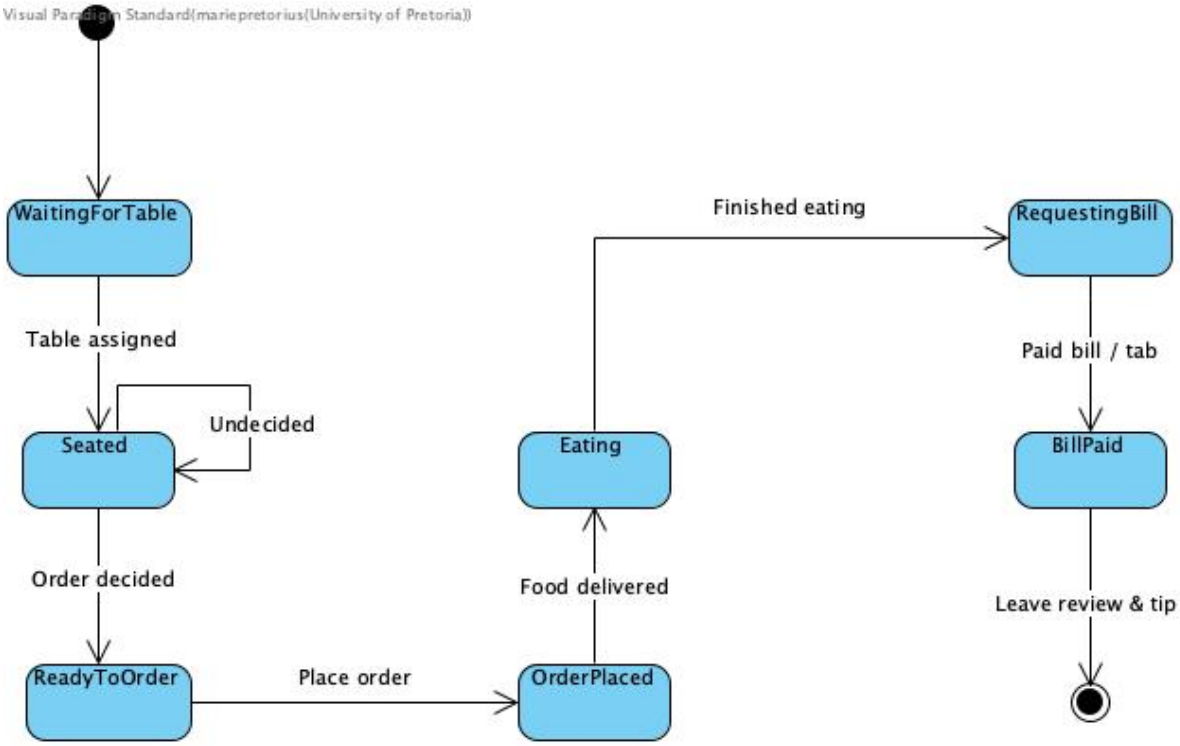
1.1 UML CLASS DIAGRAM



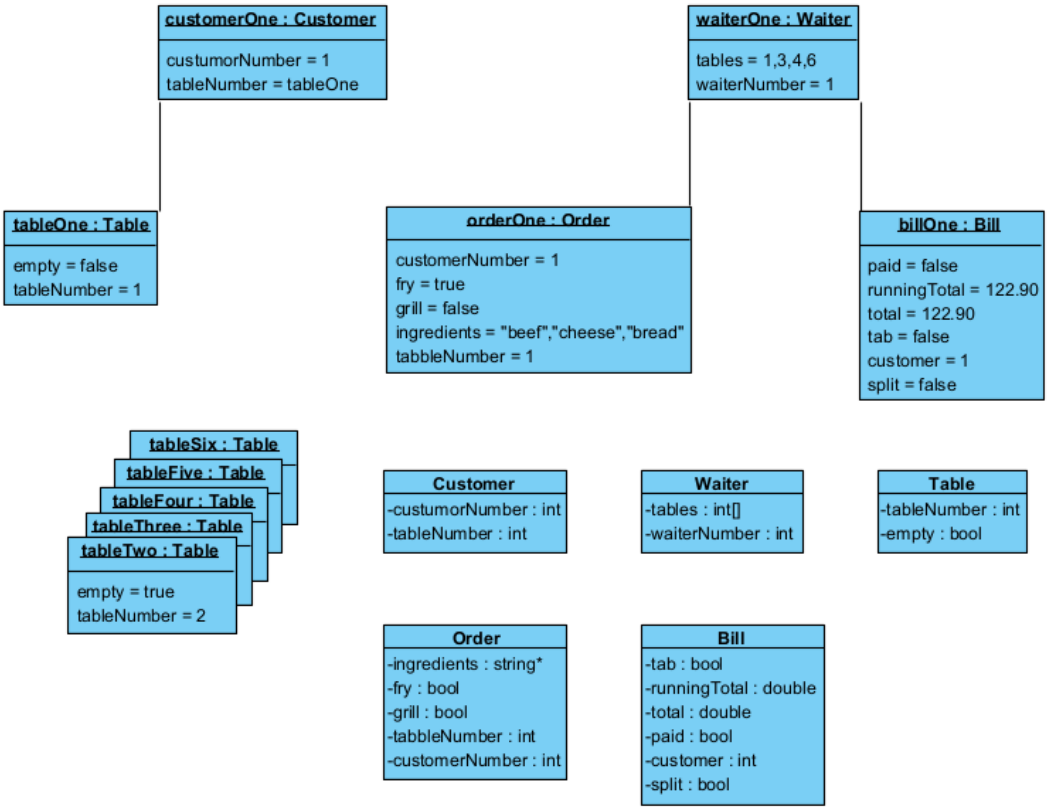
1.2 UML ACTIVITY DIAGRAM



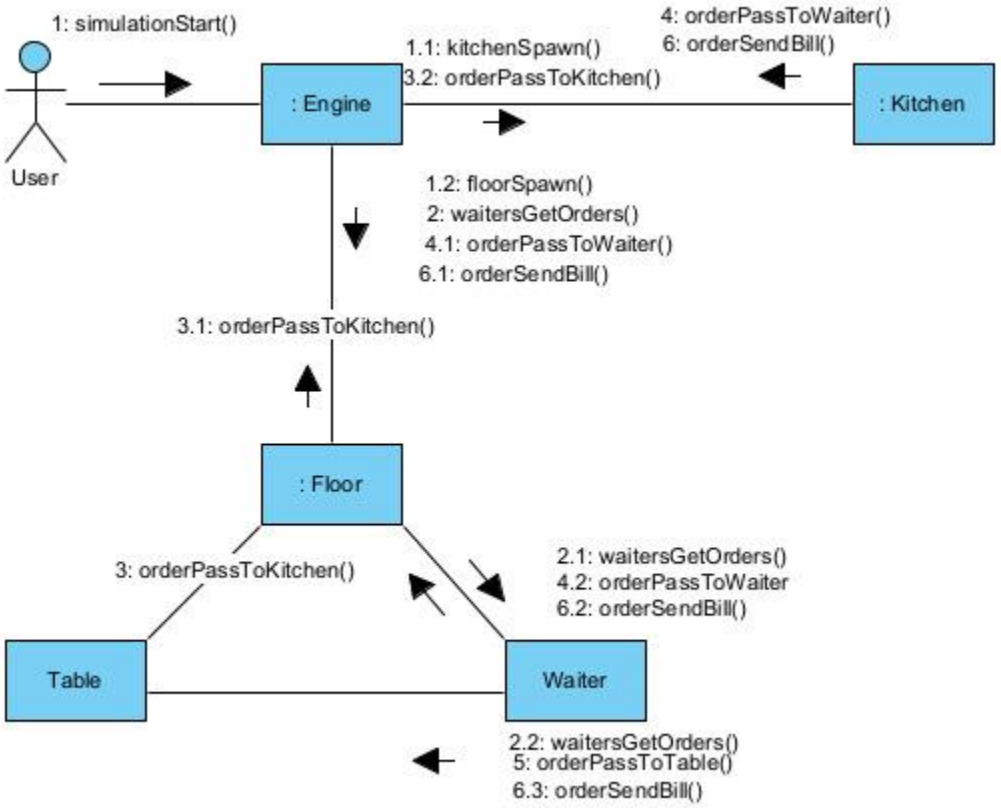
1.3 UML STATE DIAGRAM



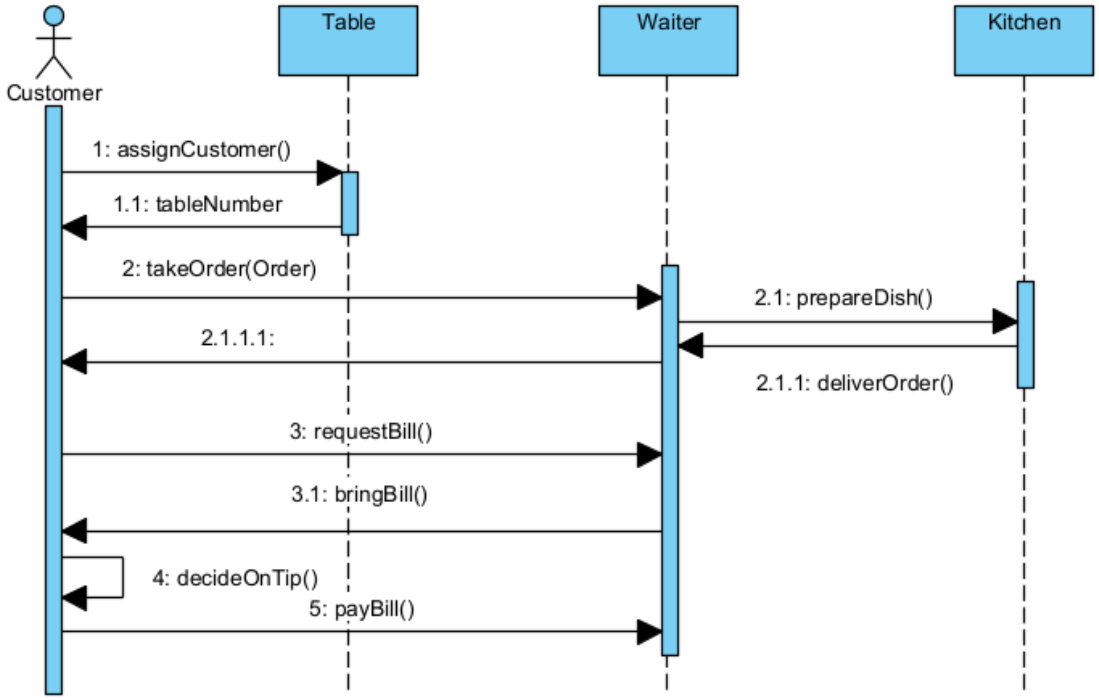
1.4 UML OBJECT DIAGRAM



1.5 UML COMMUNICATION DIAGRAM



1.6 UML Sequence Diagram



1.7 Research

1.7.1 Coding Standards

Coding standards are a set of guidelines and best practices to help developers follow code when working on a project. There are a few common practices and standards that developers should follow. Some include:

- Keeping code maintainable
- Keeping code transparent, sane and readable
- Keeping code scalable

There are a few guides that show common standards for different languages. As we will be using C++, we will specifically use C++ standards.

[https://www.st-andrews.ac.uk/digital-standards/code-standards/]

The following lists a few common practices to keep in mind when writing C++ code:

- Compile cleanly at high warning levels
    - Understand your code and understand each warning. Change your code to get rid of warnings, if necessary.
- Use an automated build system
    - Use a fully automated build system that builds the project without user intervention
- Use version control (git)

- Review each other's code
- Correctness, simplicity and clarity
    - KISS (Keep It Simple Software). Correct is better than fast and simple is better than complex, while clear is better than "cute"
- Do not optimize code prematurely
    - Measure twice, optimize once. Optimize code once the code is written and it works.
- Minimize global data (global variables)
    - Shared data increases contention and coupling, which reduces maintainability of code.
- Hide information
    - Do not expose private information of a class
- Use const proactively
    - Unchangeable values are easier to understand, track and reason about. Use constants whenever you are able to.
- Avoid magic numbers
    - Literal constants like "42" and "3.14159" are not self-explanatory and they add hard-to-detect forms of duplication which causes complication in code maintainability.
- Always initialize variables.
    - Initialize variables upon definition as uninitialized variables are a common source of bugs and errors in C++.
- Avoid deep nesting of functions.
    - Failure of giving one function cohesive responsibility will lead to long functions and nested code blocks. It is better to keep code short than long.
- Make header files self-sufficient.
    - Make sure each header file is compilable, stand-alone.
- Take parameters appropriately by value or by reference.
- Prefer canonical form of ++ and - - and prefer calling the prefix forms.
    - Prefer to use prefix form of these operators when you do not need the original value.

[https://micro-os-plus.github.io/develop/sutter-101/]

The following explain how to keep code clean and readable in C++:

Inline functions:
Define methods inline if they are minimal (10 lines or less). Inline functions can raise or decrease the size of code. Inlining a tiny accessor function will, most likely, reduce code size. Inlining a large function can increase the size of code substantially.

Namespaces:
- Namespaces contents are not indented.
- Put the namespace as '{' on the same line as the keyword's namespace.
- Place '}' on its own line
- Place the comment at the end of the namespace curly

Class definition:
- There are three categories for the class sections:
    - public
    - protected

    - Private
- Each category is indented 4 spaces.
- There should be no indentation between the public, protected and private.
- There should be no spaces between the ":" and the BaseClassName, for example: class myClass: public otherClass.
- There should be proper spacing between the colon and access control keywords, as demonstrated in previous example.
- In the case where the class name and subclass name do not fit on the same line, break the lineup.

[https://www.scaler.com/topics/cpp/cpp-coding-standards/]

1.7.2 Naming Standards

Commonly used naming standards in Software Development:

- **CamelCase**: This convention capitalizes the first letter of each word except the first one, and there are no spaces or punctuation marks between words. It is commonly used for naming variables and functions in languages like Java, JavaScript, and C#.

- Example: myVariableName, calculateTotalPrice() (Google)

- **PascalCase**: Similar to CamelCase, PascalCase capitalize the first letter of each word, but it starts with an uppercase letter. It is often used for naming classes and types.

- Example: MyClass, CustomerDetails (Martin)

- **snake_case**: In this convention, words are separated by underscores, and all letters are lowercase. It is commonly used in Python for variable and function names.

- Example: my_variable_name, calculate_total_price() (PEP 8 - Style Guide for Python Code, 2023)

- **kebab-case (or lisp-case)**: Similar to snake_case, this convention separates words with hyphens. It is often used in URLs, HTML attributes, and some CSS.

- Example: my-variable-name, data-list (W3Schools, 2023) (Chowdhury, 2022)

- **UPPER_CASE (or SCREAMING_SNAKE_CASE)**: All letters are capitalised, and words are separated by underscores. It is commonly used for constants and enum values.

- Example: MAX_VALUE, ENUM_STATUS_ACTIVE (W3Schools, 2023)

- **Abbreviations**: If using abbreviations, it's important to be consistent in how you represent them. For example, "HTML" could be written as "HTML" or "html" depending on the convention used.

- Example: HTMLParser, xmlDocument

- **Hungarian Notation**: This naming convention prefixes names with short codes indicating the data type or purpose of the variable. It's less common today but may still be found in legacy code.

- Example: strName (string), intCount (integer)

- **Domain-Specific Conventions**: Some domains or industries have specific naming conventions. For example, medical software might use specific medical terminology in variable names.

- Example: patientBloodPressure

- **Package/Directory Naming**: For organising files in a directory structure, use a consistent convention. Common conventions include lowercase with underscores or lowercase with hyphens.

- Example: my_package, my-directory (PLURALSIGHT, 2023)

- **Database Naming Conventions**: For database objects like tables and columns, naming conventions like snake_case or PascalCase are common. Prefixes for tables, like "tbl_" or "tbl", are also used for clarity.

- Example: user_table, user_id

1.7.3 Git Standards

Website from : https://medium.com/@fatihsevencan/git-commit-standards-d76f2f5f5c7f
Website accessed on 17/10/2023
Information:
NB - Use a '#' before your commit **comments**-comments could be perceived as dangerous by bots.
What commit messages to use when:

- Feat - when a new feature has been added
- Fix - when a bug has been fixed
- Refactor - when rearranging the file system/ existing code
- Docs - when making changes to documentation
- Style - when changing the format/style of your code
- Test - when adding a unit test
- Chore - changes in project organization/ configuration files.

Add a scope to the commit message in brackets() - a scope is the area that has been affected by the code change being committed. Must be in camelCase - ideally one word.

When resolving merge conflicts the commit message should be "fix(conflict):Resolve Conflicts".

Website from: https://blog.carlmjohnson.net/post/2018/git-gud/

Website accessed on:17/10/2023
Information:
Use branches and merges for development. DO NOT work in the main branch!

DO NOT push directly onto the main branch- force push to the main branch should be disabled.

Naming of branches:

- Developer_name/feature working on

A good commit is atomic - meaning your code should work when you commit- take it
From one working state to another. This helps reviewers understand what changed.

AVOID committing all changes on your repository in order to avoid committing garbage
Files- only commit the file changes that are needed.

Comments in code should be used to explain the why instead of the what! Don't explain what you are doing, but explain why you are doing it!! Just to make future
Development easier :)

Make sure to review pull requests - to ensure no known bugs can enter unseen.

NB : be mindful of people's emotions when reviewing their code. Use encouraging
Language. Use questions instead of comments. Example :"Is there a reason
why you did this and not that. My personal opinion is that doing it in that way might be
Advantageous, but I might be understanding the code/instruction wrong." Don't make the
Author of the code feels like you are just criticizing their code. Leave helpful and upbuilding
Comments.

Starting points for reviews:

- Is this commit atomic?
- Is there code that is being repeated - could we add another design pattern to avoid This?
- Is the code readable and can you understand it easily?
- Are there any visible bugs? Any safety concerns?
- Does the code follow our coding and naming standards?

If you want comments on a branch that you are still busy working on add "WIP"(work
In progress) to the Pull request- this will allow reviewers to help with a specific concern instead of
focusing on the code as a whole. If you don't want your branch to be merged
add "DNM"(Do not merge) to the pull request.

When every test has been passed and the pull request got positive reviews, allow the
Author of the code to do the merging into the main branch, they know their code the
Best and they might remember a bug/ issue they forgot about.

Website from: https://docs.github.com/en/get-started/quickstart/github-flow

Website accessed on: 17/10/2023
Information:
Branches:

- A branch is a space to work on code without changing the code in the main branch
- In a branch you can revert the changes you have made if you find that you made a mistake.
- You can push changes to a specific branch, without changing the main branch.
- When you want your final , thoroughly tested, code on the main branch, Make use of a merge request.

Commits should ideally deal with one specific feature/ issue so that you can revert your changes

Pull requests:
Pull requests are used to allow other developers to comment on your code.
Creating a pull request:

- Add a summary of your changes and explain what they do.
- If the pull request is solving an issue, link the issue to the pull request
- Add comments to specific lines of the pull request to point a reviewer's attention to it.
- You can @ someone specific.

Merging pull requests:

- ONLY merge when your pull request is approved, this will add your code to the Main branch
- After a successful merge, delete the branch to avoid anyone from working on this old branch(you can always revert your pull request/restore the branch)

Common Git Commands

Repository initialization

- `git init` : initialize a new Git repository in the current directory

Cloning

- `git clone <repository_url>` : clone a remote repository to your local machine

Branching

- `git branch` : list all branches in the repository
- `git branch <branch_name>` : create a new branch
- `git checkout <branch_name>` : switch to a different branch

Committing

- `git add <file>` : add changes to the staging area
- `git commit -m "Commit message"` : commit staged changes with a message

Merging

- `git merge <branch_name>` : merge changes from another branch into the current branch

Pull requests

- Pull requests can only be made on the repository platform (GitHub)

Viewing information

- `git status` : show the current status of the repository
- `git log` : view the commit history

- `git diff` : show the changes between commits, branches, ets.

2 REFERENCES

"Code standards - Digital standards." *University of St Andrews*, https://www.st-andrews.ac.uk/digital-standards/code-standards/. Accessed 18 October 2023.

Ionescu, Liviu. "C++ Coding Standards: 101 Rules, Guidelines, and Best Practices." *µOS++*, 22 July 2023, https://micro-os-plus.github.io/develop/sutter-101/. Accessed 18 October 2023.

Narang, Prateek. "C++ Coding Standards." *Scaler*, https://www.scaler.com/topics/cpp/cpp-coding-standards/. Accessed 18 October 2023.

Chowdhury, F. H. (2022, 8 22). *freeCodeCamp*. Retrieved from freeCodeCamp: https://www.freecodecamp.org/news/programming-naming-conventions-explained/

Google. (n.d.). *Google Java Style Guide.* Retrieved from GitHub: https://google.github.io/styleguide/javaguide.html

Martin, R. C. (n.d.). *Clean Code: A Handbook of Agile Software Craftsmanship.*

McConnell, S. (n.d.). *Code Complete.*

PEP 8 - Style Guide for Python Code. (2023, 10 11). Retrieved from Python: https://peps.python.org/pep-0008/

PLURALSIGHT. (2023, 1 15). Retrieved from PLURALSIGHT: https://www.pluralsight.com/blog/software-development/programming-naming-conventions-explained

W3Schools. (2023). *W3Schools*. Retrieved from W3Schools: https://www.w3schools.com/

https://blog.carlmjohnson.net/post/2018/git-gud/

GitHub. (n.d.). *GitHub flow*. Retrieved from GitHub.

Sevencan, F. (2023, 7 10). *Git -Commit Standards*. Retrieved from Medium: https://medium.com/@fatihsevencan/git-commit-standards-d76f2f5f5c7f