SCHOOL OF ENGINEERING AND TECHNOLOGY

COURSEWORK FOR THE
BSC (HONS) INFORMATION TECHNOLOGY
BSC (HONS) COMPUTER SCIENCE
BSC (HONS) INFORMATION TECHNOLOGY (COMPUTER NETWORKING AND SECURITY)
BSC (HONS) SOFTWARE ENGINEERING

ACADEMIC SESSION 2025; SEMESTER 2,3

PRG1203: OBJECT ORIENTED PROGRAMMING FUNDAMENTALS

DEADLINE: 8 AUGUST 2025 11:59PM (Friday)

TITLE OF THE REPORT:

BUILD A JAVA CONSOLE-BASED POKÉMON GA-OLE GAME

Academic Honesty Acknowledgement
"I ...Tiffany Fam Kar Ying…, …Lim Feng Wei…, …Sian Yi Xuan…, …Tan Wei Ting…, …Tan Wen Xi…, …Teoh En Yi… **(student name). verify that this paper contains entirely my own work. I have not consulted with any outside person or materials other than what was specified (an interviewee, for example) in the assignment or the syllabus requirements. Further, I have not copied or inadvertently copied ideas, sentences, or paragraphs from another student. I realize the penalties *(refer student handbook undergraduate programme)* for any kind of copying or collaboration on any assignment."**

Group Number: 27

Team Members:

No	Name	Student ID
1	Tiffany Fam Kar Ying (BIT) - Leader	23052301
2	Lim Feng Wei (BSC)	23062334
3	Sian Yi Xuan (BCS)	23099609
4	Tan Wei Ting (BIT)	23094709
5	Tan Wen Xi (BIT)	23093495
6	Teoh En Yi (BCS)	23051105

Prepared for: Ms. Lim Woan Ning

Table of Content

1.0 Introduction

Games development has become very crucial as part of the programs in education with the development process providing practical in learning the essences of program development. This project is aimed at the development of a simulator of Ga-Ole battle with a Pokémon based on a console, written as a part of the PRG1203 Object-Oriented Programming (OOP) Fundamentals module. This project takes advantage of the main OOP concepts, including inheritance, abstraction, encapsulation, and polymorphism, in order to make a scalable and modular game. Being written in Java and performed in a text environment with the use of a command line, this game will both keep the users entertained and provide a testbed where we display our grasp of these important concepts in programming.

The game begins by prompting the player to enter the name of their trainer, allowing them to personalize their experience in the Pokémon world. After entering the name, a Pokémon banner is displayed to set the tone for the game. The next step involves presenting the player with a menu featuring 11 gameplay options, through which they can navigate various aspects of the game.

The basics of the game are founded on many aspects such as catching of Pokémon, their battles and evolution. The catching mechanic is also carried out using Pokeball.java, which has declared four classes of Poké Balls namely, Poke Ball, Great Ball, Ultra Ball, and Master Ball. These are the sub-classes of Pokeball.java class, and they vary in their catch rates, which offers players less or greater strategic options when catching wild Pokémon. The Pokémon are captured and after being caught, they are included in the collection of the player, which is handled by the class, Player.java. The Player.java class deals with information about the player, and their Pokémon collection can be seen with such information as level, health points, type, and development prospects.

The most fundamental part of the game is the battles; these are controlled by the WildBattle.java and RivalBattle.java classes which control battles one turn at a time between the Pokémon that the player controls and either a wild or rival Pokémon. The Battle system together with the TypeEffectiveness.java calculate the damages using the Pokémon types so that we are sure in regard on combat, the types of advantages and disadvantages would be considered. This helps players to be strategic in planning their Pokémon and their moves, effectively to win.

The process of evolution is carried within the mechanism of the EvolutionData.java class, which monitors the status of evolution of each of the Pokémon's. A Pokémon develops into a stronger form when one meets specific requirements of level or other circumstances. This system brings complexity to this game because the players have the choice to ascend levels and thus making them stronger and increasing their chances of surviving battles with their Pokémon.

The Player.java class has also a method which heals the Pokémon to ensure that they are ready to face other challenges in the future by healing their HP.

The progress of the players needs to be stored permanently, and this process is facilitated through DataLoader.java and PlayerStorageManager.java. These are the classes that handle the reading and writing of data about players to three important text files namely, the players_data.txt, top_scores.txt and trainer_status.txt. The remaining data is used to describe the player and his Pokémon collection, from stats and evolution to the progress. This data is written to the file players_data.txt. Top_scores.txt monitors the High Scores of the Top 5 player scores, which the players will be motivated to get a higher score so that they can compete to have a higher ranking. The trainer_status.txt file also stores some of the progress of the player like their best Pokémon and what achievements they have attained.

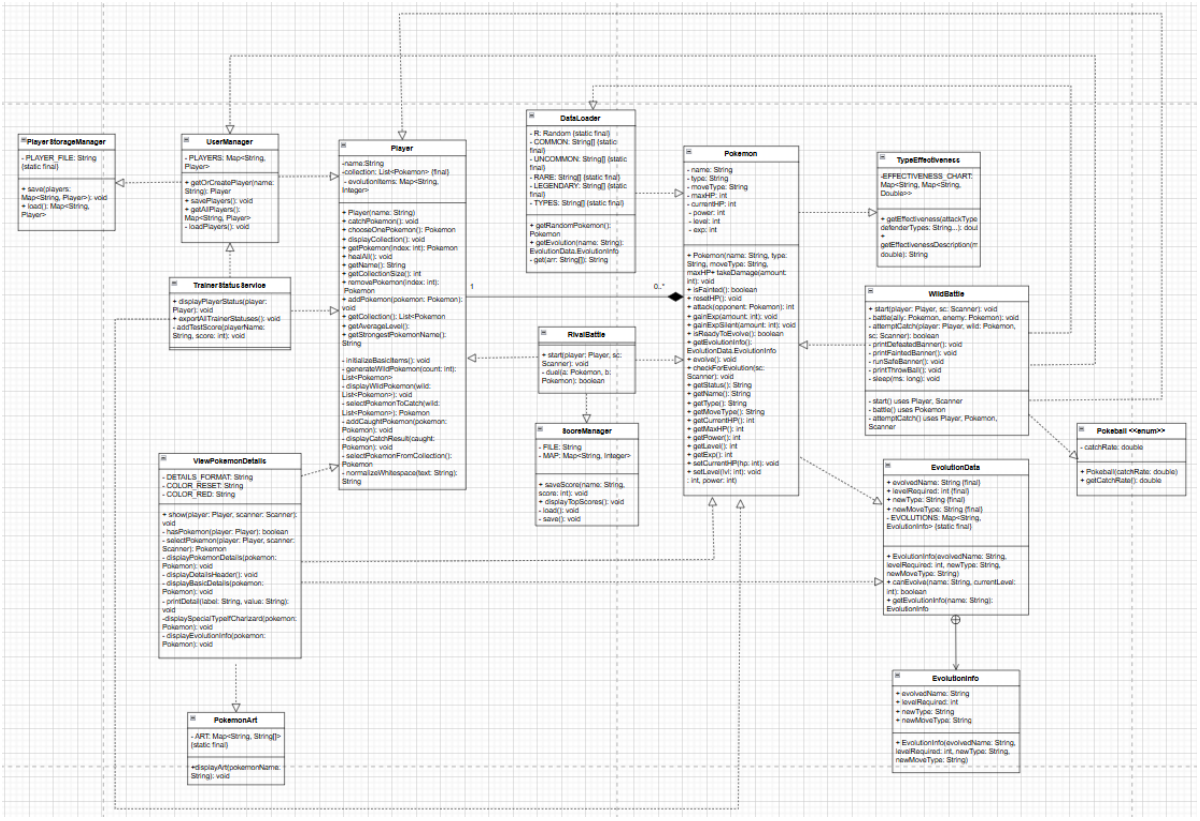
The architecture of the game is in a modular manner where each class in Java performs a particular functionality. Game initialization, user input, and game stages transition are facilitated by the Main.java class which serves as the entrance point of the game. The ScoreManager.java class does the management of the top 5 scores and the storage of the top scores in the top_scores.txt. The TrainerStatusService.java class follows the achievements in general including the highest-level Pokémon of the player and other achievements. The UserManager.java is a class that takes care of the interactions between the user that gives a hassle-free navigation experience through the menus of the game.

The game has one of its unique features, which is the ViewPokemonDetails.java class, by which a player can see detailed Pokémon information. This feature is available to players so that they can view stats, moves, health and evolutions of their Pokémon in order to make better decisions regarding their team. The Pokemon.java class handles the individual Pokémon record, and takes care of the data, their stat, moves and evolution. This is an assurance that an individual Pokémon will act appropriately in battles and also relate with the systems of the game appropriately. PokemonArt.java class brings an artistic element to the game with the presentation of ASCII art of the Pokémon. This class is an improvement to the text-based interface as it gives the player a graphic view of the Pokémon and thus enjoying and experiencing it more. The final option, option 11, allows the player to save their progress and exit the game.

To summarize this project work, it presented a great chance to use OOP concepts in the design of an interactive game. We learned how to cooperate, design in object-oriented logic and game mechanics with this project. The output is a stable Java application that illustrates our familiarity with the concept of OOPs, but it is equally captivating and pleasurable to use the application. Persistent data storage is contributed by players_data.txt, top_scores.txt and trainer_status.txt which will enable a player to resume progress and achievement over a subsequent session ensuring that the game is immersive, replay able and entertaining.

2.0 UML Class Diagram & Relationships

2.1 UML Class and Relationship Diagram



2.2 Class Relationship Justification

Inheritance

One of the most important principles of OOP is inheritance which means that it is possible to design a new class which is derived of another one. Inheritance in the Pokémon Ga-Ole Game makes it so that classes with similar characteristics or behaviours are determined and the code does not have to be duplicated, instead it can be reused. It is very much so when it comes to the following illustrations:

1) **Pokeball Subclasses:**

   The Pokeball.java class is an abstract base class that defines common attributes and methods for all types of Poké Balls in the game. The subclasses Poke Ball (0.5), Great Ball (0.75), Ultra Ball (0.85), and Master Ball (1.0) inherit from this base class, with each subclass implementing specific catch rates for their respective Poké Balls. This design promotes code reuse, as all Poké Ball types share the same basic structure but

vary in their properties (catch rate). In this case, inheritance enables different types of Poké Balls to maintain a consistent interface while allowing for customization based on catch rates.

2) **Pokémon Types:**

The Pokemon.java class is a general class to give general attributes of a Pokémon (e.g. health, level, moves). The Pokemon.java can be inherited by specific types of Pokémon and extends it with type-specific attributes or behaviour. As an example, dedicated subclasses such as Fire Pokémon, Water Pokémon, and Grass Pokémon. The subclasses may have a distinct behaviour or interactions as well (e.g., conferring extra moves (e.g., Fire Blast on a Fire based Pokémon) or changing stats). Using inheritance, the common interface and functionality specified by the Pokemon.java class is shared by each of the Pokémon types with specific behaviours according to the type of Pokémon.

Inheritance helps us eliminate repetition of code and to update and maintain more easily. New types of Pokémon, a new type of Poké Balls or even introductions of new battle mechanics can be added without having to modify the current code structure therefore making the game extensible.

Aggregation

In OOP, aggregation plays a key role as this is a relationship between objects that can be described as a “has-a” relationship. In aggregation, an object has references to an object, but the contained objects can exist in the absence of the containing object. The aggregate relationship is loosely coupled objects; hence, it is preferential in the Pokémon Ga-Ole Game. Some examples of these are:

1) **Player and Pokémon:**

The object of the java class Player.java represents aggregation of Pokemon.java objects. A player possesses a set of Pokémon, but the objects can be Pokémon even without a player. This is illustrated by the example that the Pokémon are able to be passed or released into different players collections or be taken out of their collection without pointing a player object. Such a compilation means the Pokémon collection made by the player may evolve or increase in time and Pokémon can be treated without the player.

2) **Other Services and Main.java:**

The Main.java file contains the task to control the game overall flow, to create the game and to process the user action. Main.java has some services that give important game

functionality including ScoreManager.java, PlayerStorageManager.java, and TrainerStatusService.java. Such services handle the side of the work that includes the scores of the players, data saving and navigation towards the progress of the player.

- ScoreManager. java keeps the record of the scores of the player and retains the top 5 scores in top_scores.txt file.

- PlayerStorageManager.java implements reading and writing a player data, thus the player can save and load his or her progress in the game.

- The TrainerStatusService.java could keep record of a player's milestones and achievements including the most powerful Pokémon and the general development.

The Main.java and these services example are relationships in which aggregation holds since Main.java refers to the services and the services can occur independently. This weak connection between the components enables the game to be modular, and easier to maintain. The services handle a particular task and can be upgraded or swapped and the remaining part of the game is not affected much.

3) **Pokémon and Trainer Status:**

The TrainerStatusService.java is also aggregated to the Player.java class that monitors the progress and the achievements of the player. The TrainerStatusService.java class offers the information about the most powerful Pokémon in the collection of a player and monitors the achievements reached. The reference to such service is set in the Player.java class, which is, however, autonomous and any alterations to the information about the state do not impact the Pokémon collection of the player directly.

In short, inheritance gives code reuse and specialization a mechanism where aggregation defines the relationships between objects in which one object has references to another, but the objects within the aggregate can exist by themselves. The two OOP principles aid in making the Pokémon Ga-Ole Game to be modular, scalable, and maintainable.

3.0 Features

This Pokémon Ga-Ole game has a several features that are focused on a gameplay that is easy to be engaged and interacted with. Such characteristics are realised with a set of principles of encapsulation, inheritance and modularity of the objects. Rather, the core features, which can be observed in the existing game code, are described in the following sections.

3.1 Core Features

1) **Wild Pokémon Generation:**

Wild Pokémon are randomly generated at the beginning of the game and the player can encounter them. Such wild Pokémon are picked out of a menu that ties several types of Pokémon (Fire, Water, and Grass). This is a random encounter system that makes it interesting and unpredictable, so that every time through is a unique experience. The wild Pokémon are instantiated by the subclasses of Pokemon.java and each type of Pokémon has got its own unique attributes and moves.

2) **Catch and Collect:**

Three wild Pokémon are given to the player who can stay with one to complete his or her collection. It is a strategic aspect presented by this mechanic since the players have to decide about which Pokémon to pick that has the most value in battles. The Pokeball.java class enables the chosen Pokémon to be caught by the player once he/she wins a fight, which introduces the aspect of personal improvement and success.

3) **Turn-Based Battle System**

Our battles are turn-based, where the player's Pokémon and the opponent's Pokémon (wild or rival Pokémon) take turns attacking each other. WildBattle.java and RivalBattle.java classes deal with the different battle scenarios, and they control the Battle System. The Type Effectiveness.java class computes the damage that each move inflicts as per the type advantages and disadvantages (such as, Water-type moves are more effective against Fire-type Pokémon). They go on until one of Pokémon HP comes to zero, at that time the Pokémon loses.

4) **Types of Poké Balls:**

The player is presented with the opportunity to catch wild Pokémon after defeating them in battle. The Pokeball.java class handles the different types of Poké Balls available for catching Pokémon: Poke Ball, Great Ball, Ultra Ball, and Master Ball. Each Poké Ball type has a different catch rate, and players must choose which one to use based on the strength of the Pokémon they want to catch. The attemptCatch() method calculates the success of the catch based on the type of Poké Ball and the level of the Pokémon being caught.

5) **Battle Score Calculation:**

The ScoreManager.java class will compute a battle score after every battle regarding the performance in the battle. The number of turns to be made, the amount of health left of the Pokémon, and the efficiency of the moves made are some of the factors put into consideration in the score.

6) **Saving and Displaying Top Scores:**

The ScoreManager.java class performs this maintenance of the Top 5 High Scores that are stored in the file top_scores.txt. The score of the player after every battle is compared with the scores that are already there, and the best scores are updated. The Top 5 High Scores can be viewed by the players at the end of each fight and encourages them to increase the score and get further in the game. This promotes replay-ability and makes the players feel good when out-scoring.

3.2 Add-On Feature (Implemented)

1) **Pokémon Evolution System:**

Pokémon Evolution System, which has been executed using EvolutionData.java class, enables Pokémon's to evolve to powerful form when meets the conditions i.e. It reaches the level that could lead to any evolution. As an example, a Charmander will turn into Chameleon and then Charizard during its experience. Such development upgrades the stats of the Pokémon, opens new moves, and improves battle capabilities, applying a definite progression scenario to the players. This aspect provides an enthralling aspect to gaming because by the time a player progresses further in the game, player is rewarded with more powerful Pokémon.

2) **Viewing Pokémon Details:**

ViewPokemonDetails java file gives the players an idea about the detailed information of every Pokémon. This aspect brings important statistics like level, HP, type, and available moves of the Pokémon. It gives the players a chance to see these stats which enable them to make accurate decisions when it comes to selecting which Pokémon to use in a battle, so that they select the appropriate Pokémon to use against the opponent. This just gives an element of strategy to the game because the players could maximize the performance of the team through the detailed information given.

3) **Player Storage Manager:**

PlayerStorageManager.java class is vital in saving and loading the data of the players. It communicates with players_data.txt to store and read the collection and stats and the progress of the player during the game. This is because it enables the users to save their

game status and then resume it later so that they are not required to start again. The PlayerStorageManager.java is also efficient in the way it handles the player data such that they can have a smooth gameplay even after they have disconnected.

4) **Trainer Status:**

Trainer Status is an attribute in the TrainerStatusService.java and monitors the progress of the player through the game. It keeps track of accomplishments earned like the toughest Pokémon in the Pokémon collection of a player and other significant achievements. This information is saved in trainer_status.txt where the players can have a glimpse of their experience. This aspect encourages the players to keep going, and they could see what progress they are making as they go and desire to achieve more in the process of playing.

5) **Persistent Data Storage:**

Persistent Data Storage is such an aspect that saves on the progress of the players between two sessions. The player data including Pokémon collection, stats and battle progress are saved in players_data.txt through DataLoader.java and PlayerStorageManager.java. The reading and writing of this information occur in the PlayerStorageManager.java class which ensures that the player could resume the game that he left. Also, the progress of the player, such as his or her most powerful Pokémon, is saved in the trainer_status.txt file and the top 5 battle scores are saved in top_scores.txt. Using these files, it is easy to retrieve, and this also means that the players will not have to restart at the beginning of the game and lose their progress.

6) **Pokémon Art (ASCII Art):**

Pokémon Art (ASCII Art) is offered by the class PokemonArt.java as this increases the visual quality of the game as it illustrates Pokémon's with the ASCII art. This contributes the element of creativity in the graphics of the game, where, when players fight their Pokémon or check details, they get to see a graphical representation of their Pokémon. All the Pokémon have their very own ASCII art which makes the experience more immersive and makes it more interesting to use. This is a characteristic which brings in the character and style in the game particularly in the text site.

Our Pokémon Ga-Ole Game offers a very interesting and dynamic game play experience with some critical features including wild Pokémon battles, turn-based combats, and the evolution of the Pokémon. The game stimulates thinking and development with such features as the battle score system, the storage data permanently, and the possibility of observing the detailed stats of Pokémon. Other improvements such as ASCII art add even more to the game experience of the player making the game visually interesting to play. These characteristics together with the

application of object-oriented principles such as encapsulation and modularity make the game enjoyable and scalable, providing the game player with a contesting and satisfying experience.

4.0 Conclusion

The path to completion of the Pokémon Ga-Ole Game project was a very rewarding one and a thoroughly enriching one and we truly glad to have observed in full, the implementation of the OOP concept to a game environment. During the designing and development of this console-based simulator, the important concepts of OOP like inheritance, encapsulation, abstraction, and polymorphism have been applied to develop a modular, extensible, and supportable game. They have not only applied these concepts to the core mechanics but have used the concepts promoted by it to offer a more streamlined user experience and a living and exciting gameplay experience.

Through combination of those key elements of the gameplay, including catching Pokémon, battling, evolving them, and making certain strategic decisions, the project grants the players with the satisfying interactive experience. Catching the evolving Pokémon using different kinds of Poké Balls and turn based combats involving critical thinking and solving of problems brings about problem solving and strategic planning into the software. The system in which the battles are built around the Pokémon types and how effective they are to each other's type makes all the battles feel fresh and will need the proper choices. Moreover, the introduction of Pokémon Evolution process adds some complexity to its progression system, as one is to face new challenges and receive new rewards, further contributing to a feeling of accomplishment by the player.

The architecture of the game, resting upon the architecture of modular classes and a properly outlined structure can easily be extended later and is also maintained. Aggregation and inheritance have been successfully employed to achieve dynamic associations between objects in the sense that any new functionality may be added without conflict with the existing code base. Utilization of lasting data storage with the help of text files (players_data.txt, top_scores.txt, and trainer_status.txt) enables the users to preserve their achievements, which promotes replay and contributes to the prolonged interest in playing the game.

In addition, the addition of innovative details like the Pokémon Details Viewer that allows players to play games with comprehensive information regarding their Pokémon and the Pokémon Art feature that introduces the artistic level into the gameplay makes it even better. Such supplements add in to the fact that the interaction with the game becomes more immersive and enjoyable to the players making the game not just a technical marvel but also a sight to behold and a lovely thing to watch.

It can be concluded that the Pokémon Ga-Ole Game project can be a successful combination of the key OOP concepts with an entertaining gameplay process to appear as an interactive, strategic, and entertaining game. With this project, we have also been able to prove not only

our knowledge of OOP but also the effectiveness of object-oriented design to make a game that can be scaled and maintained easily. The functional features, artistic improvements, and a strong technical basis can make sure that the game is not only entertaining but also provides chances of improvement. The project has taught us many useful lessons as developers and has been very instrumental in our decision-making processes on how OOP could be applied in practice when writing a programme. It has also been the beginning of more projects in game development.