BLG 252E - Object Oriented Programming
Assignment #1

Due: April, 4th 23:59

*"**Jessie**: Team Rocket blasting off at the speed of light.*
***James**: Surrender now or prepare to fight.*
***Meowth**: Meowth. That's right."*

-Pokémon (1997)

Introduction

For this assignment, you are expected to implement Satoshi Tajiri's great game "Pokémon" as a text-based strategy game. For any issues regarding the assignment, you can ask your questions on Ninova Message Board. Before writing your question please check whether your question has been asked. You can also contact T.A. Barış Bilen (**bilenb20@itu.edu.tr**).

Implementation Notes

The implementation details below are included in the grading:

1. Please follow a consistent coding style (indentation, variable names, etc.) with comments otherwise you will get **minus** points.
2. You are not allowed to use STL containers.
3. You may (and should) implement any getter methods when they are needed.
4. Make sure that there is no memory leak in your code otherwise you will get **minus** points.
5. Define required functions and object references as **const** in all cases where necessary otherwise you will get **minus** points.
6. Since your code will be tested automatically, make sure that your outputs match with sample scenario for given inputs. You will be provided with a calico file to check your assignment output.

Submission Notes

1. Your program should compile and run on Linux environment. You can (and should) code and test your program on Docker's Virtual Environment. Do not use any pre-compiled header files or STL commands. Be sure that you have submitted all of your files.
2. Submissions are made through **only** the Ninova system and have a strict deadline. Assignments submitted after the deadline will not be accepted.
3. This is not a group assignment and getting involved in any kind of cheating is subject to disciplinary actions. Your assignment should not include any copy-paste material (from the Internet or from someone else's paper/thesis/project).

1 Implementation Details

You are expected to implement 4 classes; **Pokemon**, **Pokedex**, **Player**, **Enemy**.

1.1 Pokemon

Private Attributes:

Pokemon class has a **name** (eg. Pikachu) a **Health Point (HP)(eg. 100)** and a **Attack Value (Atk)(eg. 20)**.

Methods:

1. **Constructor(s):** You are expected to write 3 constructor. First one is a default constructor which will initialize all values, second one will construct a Pokemon, it will take a string for name and an integer for attack value of the Pokemon and it also has to set default Pokemon hp to 100 and third one will be a copy constructor which will be taking a Pokemon class object and copy its contents.
2. **Getter(s):** You can (and should) implement getter methods for all attributes.

Warning: If you need any other methods for Pokemon class, you are expected to implement them as an inline function.

1.2 Pokedex

Private Attributes:

Pokedex class has a **Pokedex Array** (Array That Can Hold Pokemon Class) and a int **value** (Keeps Track of the Position)

Methods:

1. **Constructor(s):** The constructor should initialize value attribute as 0.
2. **updatePokedex:** Method to add new pokemons to the pokedexArray. Only new pokemons are allowed to be added. Check for duplicates!
3. **printPokedex:** Method to print Pokemon names from pokedexArray.

1.3 Player

Private Attributes:

Player class has a **name** (eg. Ash), **Pokemon Number** (Holds Number of Pokemon's), **Pokeball Number** (Holds Number of Pokemoballs), **Badge Number** (Holds Number of Badges), Player's **Pokemon** (Object with Pokemon Class) and **Player's Pokedex** (Object with Pokedex Class).

Methods:

1. **Constructor(s):** You are expected to write 2 constructor. First one is a default constructor which will initialize all values, second one will construct a new player with a given **name** (string) and **Pokemon** (object).
2. **showPokemonNumber:** Method to return pokemonNumber.
3. **showPokeballNumber:** Method to return pokeballNumber.
4. **showBadgeNumber:** Method to return badgeNumber.

5. **getPokemon:** Method to return playerPokemon object.
6. **battleWon:** Method to update badgeNumber and pokeballNumber. Every time you win a battle your badge number goes up by 1 and your pokeBallNumber goes up by 3.
7. **catchPokemon:** Method to update pokemonNumber and pokeballNumber. Every time you catch a Pokemon your pokemonNumber goes up by 1 and your pokeballNumber goes down by 1.

1.4 Enemy

Private Attributes:

Enemy class has a **name** (eg. Misty) and enemy's **Pokemon** (Object with Pokemon Class).

Methods:

1. **Constructor(s):** You are expected to write 2 constructor. First one is a default constructor which will initialize all values, second one will construct a new enemy with a given **name** (string) and **Pokemon** (object).
2. **getPokemon:** Method to return enemyPokemon object.
3. **getName:** Method to return name of the enemy.

Notice: You are already given a skeleton code. Please examine it before writing your own codes. Skeleton code for classes (header file) includes all necessary classes and variables. You are expected to implement methods of these classes.

1.5 Necessary Methods You Need to Implement in Main.cpp:

1. **readEnemyNames:** Method to read enemyNames.txt file and create a dynamic string array for the enemy names.
2. **readPokemonNames:** Method to read pokemonNames.txt file and create a dynamic string array for the Pokemon names.
3. **characterCreate:** Method to create a character. You are expected to create a player with the chosen name and Pokemon. Returns a player class.
4. **fightEnemy:** Method for fighting with an enemy. You are expected to create a Pokemon and an enemy, update the pokedex if the created Pokemon is not in the pokedex and simulate a fight. Player has two options; either he/she can fight or runaway. Fighting will happen in turns. Every turn player will have these option. Each turn both Pokemon will decrease its opponents hp according to their attack value. First Pokemon to reach 0 or lower then 0 health will lose. Player and enemy pokemon should start from full health(100hp) every time you enter a fight.
5. **catchPokemon:** Method for catching a pokemon. You are expected to create a new Pokemon and update pokedex if the created Pokemon is not in the Pokemon. Player has two options; either he/she can catch the Pokemon or runaway.

Warning: To make grading easy set player Pokemon's atk value to 20 and hp value to 100 and set enemy Pokemon's atk value to 10 and hp value to 100. So with these values player should always win the fight in 5 turns.

Notice: Select a new pokemon and enemy every time fightEnemy or catchPokemon functions called.

Notice: All needed functions were given to you in skeleton code. You are expected to implement the methods inside these defined functions.

2 Gameplay

In this game you can fight with enemies to win badges and Pokeballs or catch Pokemons using Pokeballs to expand your Pokemon collection. The possible actions in the game are as follows:

1. **Fight for a badge:** Brings an enemy for you to fight and win a badge.
2. **Catch a Pokemon:** Brings a pokemon for you to catch.
3. **Number of Pokemons:** Shows how many Pokemons you have.
4. **Number of Pokeballs:** Shows how many Pokeballs you have.
5. **Number of Badges:** Shows how many badges you earned.
6. **Pokedex:** Shows the different pokemons that u see in your advanture.
7. **Exit:** Exits the game.

Notice: You can check the gameplay **here**.

3 How to compile, run and test your code

If you want to compile and run the provided code on terminal, you can use these commands:

g++ -Wall -Werror main.cpp pokemon.cpp pokemon.h -o assignment1
./assignment1 enemyNames.txt pokemonNames.txt

Notice: Order of the text files are important do not change it!

You can (and should) run the calico file on terminal to check your assignment with the command:

calico assignment1_test.yaml −−debug

4 Submission

Submit your **main.cpp**, **pokemon.cpp** and **pokemon.h** files to Ninova. Before submitting please make sure you are passing all cases in calico file. Calico file you received is not the grading file so taking 100 does not corresponds to full grade for your assignment. Passing all cases will show you that your output is compatible with the grading file. Otherwise you might not receive a full grade from the assignment.