Computer Science 12 **Name: _____**

Pokemon Game **Date: _____**

Create a Pokemon game using object oriented programming. Your game will simulate a pokemon match. The user will battle against the computer. **You will each start with a team of 4 pokemon**. The battle is won when either the user or computer has no pokemon left to battle.

Several csv files have been provided.

The **pokemon.csv** file contains information for each pokemon: number, name, type, hp, attack, defense.

The **moves.csv** file contains information for each move: name, type, power.

The amount of damage a move does against a defending pokemon depends on the attacking pokemon's attack, defence, the move's power and a multiplier.

$$Damage = \left(2 \times Move_{Power\ Level} \times \frac{Attacker_{Attack\ Level}}{Attacker_{Defense\ Level}} \div 50 + 2\right) \times Multiplier \times 5$$

The multiplier depends on the type of the move and the type of the defending pokemon.

The **multiplier.csv** file contains information to determine the multiplier.

Damage lowers the opponent's hp. When a pokemon's hp reaches 0 it is said to have **feinted** and can no longer be used in battle.

Game

At each turn in the game – the player must choose to Fight, Bag, Pokemon or Run

Fight

When you select fight, you will get the option of choice which move to use. After you select the specific move, damage will be calculated and applied. A summary will show

Bag

Each player's bag will be initialized to contain status condition healing items. Some examples of these items are Antidote, Awakening, Burn Heal, Full Heal, Ice Heal, Paralyze Heal, Persim Berry. Each of these items heal certain status conditions.

Item	Status Condition
Antidote	Poison
Awakening	Sleep
Burn Heal	Burn
Full Heal	All
Ice Heal	Frozen
Paralyze Heal	Paralysis
Persim Berry	Confusion

Status Condition

Certain moves cause certain status conditions. When a pokemon has a status condition they are prevented from fighting until their status condition is healed.

The **special_status_pokemon.csv** file contains information about which moves cause which status conditions.

Pokemon

When you select pokemon you can call another pokemon from you team to battle – as long as they have not feinted.

Run

The user will select run to exit the game (or play again).

GUI

Your game **can** have a GUI. You can choose to build your gui using the Jave Swing library.

HP

A pokemon's HP will be visually indicated with a health bar on the gui. Consider using a `JProgressBar` widget for the health bar.

https://docs.oracle.com/javase/tutorial/uiswing/components/progress.html

Planning

Create a UML diagram that gives an overview of each class involved and the relationship between them.

Rubric

Expectations & Achievement Categories		Level 4	Level 3	Level 2	Level 1
Communication	**Class docstrings include description and description of each instance variable.** **Method docstrings include description, description of each parameter and return.**	Docstrings are well written and clearly explain how the method is used.	Docstrings are mostly well written and mostly explain how the method is used.	Docstrings are provided but missing criteria and/or unclear.	Minimal/missing.
Knowledge	**Approaches software design using object oriented programming (as shown in UML)**	UML class diagram is clear and complete.	UML class diagram is mostly clear and complete.	UML class diagram is somewhat complete.	Missing/missing.
	Exceptions are thrown and handled effectively.	Program is robust and handles errors. Try/catch used effectively.	Program is mostly robust and mostly handles errors, try/catch mostly used appropriately.	Program is somewhat robust and somewhat handles errors, try/catch somewhat used appropriately.	Minimal/missing.

Application	**Programming structures are used effectively.**	Data structures (arrays, arraylists etc) are used effectively. Built in methods are used effectively. No unnecessary duplication of algorithms. Conditionals and loops (for and while) are used well and appropriately.	Data structures are mostly used effectively. Built in methods are mostly used effectively. Minimal duplication of algorithms. Conditionals and loops (for and while) are mostly used appropriately.	Data structures are somewhat used effectively. Built in methods are somewhat used effectively. Some duplication of algorithms and/or data structures. Conditionals and loops (for and while) are somewhat used appropriately.	Minimal/missing.
Application (5%)	**Effectively creates a functional GUI.**	Frames, panels, buttons, and labels are used effectively. User experience is smooth.	Frames, panels, buttons, and labels are mostly used effectively. User experience is mostly smooth.	Frames, panels, buttons, and labels are somewhat used effectively. User experience is somewhat smooth.	Minimal/missing.

Thinking	**Classes are designed and implemented effectively to solve complex problems.**	Design is clear and easy to understand. Program is divided into classes – each class has a singular purpose. Each class has strong cohesion – all methods in class support singular purpose. Instance variables (properties of object) are well thought out. Date is hidden. Getters and setters are implemented. Special methods are implemented effectively. Methods are written and implemented effectively with purpose. Methods are concise – no method, including the main method should exceed 30 lines.	Design is mostly clear and easy to understand. Program is mostly divided into classes where each class has a singular purpose. Each class mostly has strong cohesion. Instance variables (properties of object) are mostly well thought out. Date is hidden. Getters and setters are implemented. Special methods are implemented mostly effectively. Methods are written and implemented mostly effectively with purpose. Methods are mostly concise.	Design is somewhat clear and easy to understand. Program is somewhat divided into classes where each class has a singular purpose. Each class somewhat has strong cohesion. Instance variables (properties of object) are somewhat thought out. Date is hidden. Getters and setters are implemented. Special methods are implemented somewhat effectively. Methods are written and implemented somewhat effectively with purpose. Methods are somewhat concise.	Minimal/missing.
Thinking	**Software solution meets defined requirements.**	The program exceeds the requirements in the specification.	The program meets the requirements in the specification.	The program meets most of the requirements in the specification.	Minimal/missing.