CSE 8B Homework 8:
Implementing the Game - Pokemon Go

Learning goals:

- Write classes in an inheritance hierarchy in Java
- Override methods in Java
- Use polymorphism to call functions
- Write recursive methods

Introduction to the Pokemon game

In this assignment you will be implementing methods in a class to provide the functionality for a (simple) text-driven Pokemon Go simulator. If you are not familiar with Pokemon Go, don't worry - here's what you need to know:

Let's start with the basics. Pokemon are creatures that have a lot of different characteristics. There are different kinds of Pokemon (you can find a list here). A Pokemon has a name, a type (there are 18 different types - here's a full list) and each Pokemon makes a unique sound (most often, they just call out their own name). For example, *Pikachu* is a Pokemon that is an *Electric* type, and it says *"pika pika"*. (In the original games, each Pokemon can have either one or two types, but *for this assignment, we will assume that a Pokemon has only one type*).

The player of this game is able to encounter *"Wild"* Pokemon, and attempts to befriend them. In order to do this, the player must try to "catch" the Pokemon using a special item known as a *"Pokeball"*. Pokeballs are not guaranteed to always work. There are different kinds of Pokeballs - each one has a different performance level, and the higher the performance, the higher the chance that the Pokeball successfully captures the Pokemon. If the attempt is unsuccessful, then the player can continue to try catching the Pokemon as long as they have more Pokeballs with them. If the attempt is successful, then the Wild Pokemon joins the player as a Pal Pokemon.

There are other items that can be used to improve the player's chances of successfully capturing a Pokemon. These items are called *"Berries"* - the player can feed a Berry to a Wild Pokemon before throwing a Pokeball at it, which increases the Wild Pokemon's patience and lowers its escaping speed, thus making it more likely to get caught. Again, there are different kinds of Berries, with each Berry affecting the Wild Pokemon's patience and speed by different amounts. Note, however, that a Berry can only improve the chances of catching a Pokemon, but a Berry never worsen the chances.

The player has a Backpack which is used to contain all their items (Pokeballs and Berries). Each item is single-use (*i.e.* after using one, the item gets consumed). So, for example, if the player has 3 Pokeballs, they only get 3 attempts to catch a Pokemon. The player also has something called a *"Pokedex"*, which is a kind of e-journal that is used to maintain a record of all Pokemon (both Wild and Pal) that the player has encountered.

Before you start: Download and Understand the Starter Code

In this PA, you will start with some code that we have written for you. Please download the starter code provided here: PA8 Starter Files

This is what your file structure for the entire PA8 should look like:

```
+-- PA8/
|     +--  Item.java              Don't edit this file
|     +--  Pokemon.java           Don't edit this file
|     +--  Pokeball.java          Create and edit this file
|     +--  Berry.java             Create and edit this file
|     +--  PalPokemon.java        Create and edit this file
|     +--  WildPokemon.java       Create and edit this file
|     +--  Backpack.java          Edit this file
|     +--  Pokedex.java           Edit this file
|     +--  PA8Tester.java         Edit this file
|     +--  RecursionWarmup.java   Create and edit this file
```

It is very important to organize the files like this in order for the provided Library methods to work correctly. You will run `javac` and `java` from within the PA8 directory.

Here is a description of each file in your `PA8` directory. **DO NOT MODIFY the following files:**

- `Item.java`: A parent class (superclass) of `Berry.java` and `Pokeball.java`. It has the basic `name` of the item.
- `Pokemon.java`: A parent class (superclass) of `PalPokemon.java` and `WildPokemon.java`. It has the basic `name`, `sound` and `type` information of a Pokemon.

CREATE and EDIT the following files:

- `Pokeball.java`: A class that inherits from `Item.java`. This represents the Pokeball that is used to capture WildPokemons.
- `Berry.java`: A class that inherits from `Item.java`. This represents the berry that can increase the chance of catching the WildPokemons.
- `PalPokemon.java`: A class that inherits from `Pokemon.java`. This represents the Pokemons you caught. The pokeball that was used to capture this Pokemon would be recorded.

- `WildPokemon.java`: A class that inherits from `Pokemon.java`. This represents a Pokemon you encounter for the first time. It has no trainers. By chance, you can catch it and be the trainer. Then, they would become PalPokemons.

EDIT the following files:

- `Backpack.java`: A class that is an inventory for all the items. It has one ArrayList for items (Berry and Pokeball). You can display what is in the backpack with the `display()` method.
- `Pokedex.java`: A class that is a wikipedia for Pokemons that you have encountered. It has one ArrayList for Pokemons (PalPokemon and WildPokemon). You can display what is in the Pokedex with `display()` method
- `PA8Tester.java`: You should write your tests in this folder. Instructions explained separately in each part.

CREATE and EDIT the following file:

- `RecursionWarmup.java`: This class will contain the methods from Part 7.

Reminder: Coding Style (0.5 points)

Don't forget to adhere to our CSE 8B coding style:

https://cseweb.ucsd.edu/classes/fa20/cse8B-a/styleguide.html

PLEASE BE SURE that you remove the TODO comments in the code after you are done with each part!

Part 1: `Pokeball.java` (creating and testing) (Pair programming allowed) (1.5 points)

Create a class named `Pokeball` that extends the `Item` class in a file named `Pokeball.java`.

This class has one *private* member variable:

- `int performance // This field will do with the catch rate.`
  `// Higher performance will increase the catch`
  `// rate more.`

This class should contain the following *public* methods. **You must name your methods *exactly* as specified below:**

Two constructors:

```
public Pokeball ()
```

This constructor should call the parent class default constructor and initialize the member variable value to `0`.

`public Pokeball(String pokeballName, int performanceIn)`
This constructor should call the parent class constructor to set the member variable `name` of the parent class to `pokeballName`. Also, it should initialize the member variable `performance` to the value of the parameter `performanceIn`.

One getter method:
`public int getPerformance()`
This method returns the value of the performance variable.

Override `toString` method:
`public String toString()`
Override the toString method so it returns a string representation of the object. Return a string that represents the `name` of the pokeball and its `performance` (new-line separated). For example:

```
Great Ball
performance: 10
```

Test `Pokeball.java` in `PA8Tester.java`

1. Create 2 Pokeball objects with different names and performances
2. Print the `name` and `performance` of the two objects on a separate line using the getter method. (we are going to add tests for `toString` method in Part 5)

Below are three kinds of balls you might choose to create in your tests:

Pokeball Name	performance
Regular Ball	0
Great Ball	10
Ultra Ball	30

Part 2: `Berry.java` (creating and testing) (Pair programming allowed) (1.5 points)

Create a subclass named Berry that extends Item class in a file named `Berry.java`.

This class has two *private* member variables:

- `int patienceIncrement // This field will also help the catch rate.`
  `// Higher patienceIncrement will increase catch rate more.`

- `int speedDecrement // This field will help the catch rate from another perspective. Higher speedDecrement will decrease the moving speed of a wild Pokemon to make easier for a Pokeball to hit.`

This class should contain the following *public* methods. **You must name your methods *exactly* as specified below:**

Two constructors:

`public Berry ()`

This constructor should call the parent class default constructor and initialize both of the member variable values to `0` as default.

`public Berry (String berryName, int patienceInc, int speedDec)`

This constructor should call the parent class constructor to set the member variable `name` of the parent class to `berryName`. Also, it should initialize the member variable `patienceIncrement` to the value of the parameter `patienceInc` and `speedDecrement` to `speedDec`.

Two getter methods:

`public int getPatienceIncrement()`

This method returns the `patienceIncrement`.

`public int getSpeedDecrement()`

This method returns the `speedDecrement`.

Override `toString` method:

`public String toString()`

Override the toString method so it returns a string representation of the object. Return a string that represents the `name` of the berry, `patienceIncrement`, and `speedDecrement`. For example,

```
Razz Berry
patienceIncrement: 10
speedDecrement: 0
```

Test `Berry.java` in `PA8Tester.java`

1. Create 2 berry objects with different `name`s, `patienceIncrement`, and `speedDecrement`.
2. Print the `name`, `patienceIncrement`, and `speedDecrement` of each berry on a separate line using getter methods. (we are going to add tests for `toString` method in Part 5)

Below are three kinds of berries you might choose to create:

Berry Name	patienceIncrement	speedDecrement
Razz Berry	10	0
Nanap Berry	0	10
Golden Razz Berry	30	0

Part 3: `PalPokemon.java` (creating and testing) (Pair programming allowed) (2 points)

Create a subclass named PalPokemon that extends the Pokemon class in a file named `PalPokemon.java`.

This class has one *private* member variable:

- `String pokeballName`

PalPokemon represents a pokemon you caught. PalPokemon has a String `pokeballName` field. The name of the ball that caught the pokemon will be recorded here. This class should contain the following *public* methods. You must name your methods *exactly* as specified below:

Two constructors:

`public PalPokemon ()`

This constructor should call the parent class default constructor and initialize the member variable. Set the `pokeballName` to be `"undefined"`.

`public PalPokemon (String pokemonName, String pokemonSound, String pokemonType, String pokeballName)`

This constructor should call the parent class constructor to set `name` of the parent class to `pokemonName`, `sound` of the parent class to `pokemonSound`, and `type` of the parent class to `pokemonType`. Also, initialize the member variable `pokeballName` to the value of the parameter `pokeballName`.

One getter method:

`public String getPokeballName()`

This method returns the `name` of the Pokeball (NOT the Pokemon).

One printing method:

`public void comesOutFromBall()`

This method will print this Pokeball's information.

`[Pokemon name] in [Pokeball name], [Pokemon type] type pokemon.`

Pokemon name and type is from the parent class Pokemon, and Pokeball name is from PalPokemon. Then, it will use the parent class method `speak()` to make your pokemon speak. For example:

```
Pikachu in ultraball, electric type pokemon.
Pikachu: pikapika!
```

Override `toString` method:

`public String toString()`

Override the toString method so it returns a string representation of the object. Return a string that represents the name of the pal pokemon, specify that it's a pal pokemon, pokeball name and the type of the pokemon. For example,

```
Squirtle, PalPokemon
pokeballName: superball
type: water
```

Test PalPokemon.java in PA8Tester.java

1. Create 2 PalPokemon objects with different Pokemon `name`s, `sound`s, `type`s, and Pokeball `name`s
2. Print the `name`, `sound`, `type`, and `pokeballName` of each object on a separate line using getter methods.
3. Make a function call to `comesOutFromBall()` for each object (we are going to add tests for `toString` method in Part 5)

Below are five kinds of pokemons you might choose to use in your tests:

Pokemon Name	Sound	Type
Pikachu	pikapika	electric
Bulbasaur	bulb	grass
Charmander	char	fire

Squirtle	squir	water
Mew	mew	psychic

Part 4: `WildPokemon.java` (creating and testing) (Pair programming allowed) (2 points)

Create a subclass named WildPokemon that extends the Pokemon class in a file named `WildPokemon.java`.

This class has three *private* member variables:

- **`int patience`**: An integer representing the patience of a wild pokemon. With no other effects, *we assume a wild pokemon's patience is from 0 to 100 inclusively.* Wild pokemons with high patience are easier to catch, and vice versa. If the patience is 100, then your catch will definitely succeed. If the `patience` is 0, then the wild pokemon will disappear.

- **`int speed`**: An integer representing the move speed of a wild pokemon. With no other effects, *we assume a wild pokemon's speed is from 0 to 100 inclusively.* When we throw a pokeball to catch a wild pokemon, we will also have a speed for the ball (ballSpeed). The ball will only hit the wild pokemon if `ballSpeed` is faster than or equal to the `speed` of the pokemon. Remember, there is no `ballSpeed` field for Pokeball class. We will explain how to get `ballSpeed` later.

- **`int timesEscapedFromBall`**: After your ball hits a wild pokemon, there will be two outcomes: 1. The Pokemon is caught, or 2. the Pokemon escapes from the ball. This field records the times that a wild Pokemon escapes from a Pokeball. It should be initialized to `0` when you create any new wild pokemons.

This class should contain the following *public* methods. You must name your methods *exactly* as specified below:

Two constructors:

```
public WildPokemon ()
```

This constructor should call the parent class default constructor and initialize the member variable. Set the `patience` to be `100`, `speed` as `0`, and `timesEscapedFromBall` as `0` as well.

```
public WildPokemon (String pokemonName, String pokemonSound, String
pokemonType, int patienceIn, int speedIn)
```

This constructor should call the parent class constructor to set the `name` of the parent class to `pokemonName`, `sound` of the parent class to `pokemonSound`, and `type` of the parent class to `pokemonType`. Also, initialize the member variables `patience` as `patienceIn`, `speed` as `speedIn`, and `timesEscapedFromBall` as 0.

Three getter methods:

`public int getPatience()`
This method returns the `patience` of the WildPokemon.

`public int getSpeed()`
This method returns the `speed` of the WildPokemon.

`public int getTimesEscapedFromBall()`
This method returns the number of times escaped from the ball of the WildPokemon.

Three setter methods:

`public void setPatience(int newPatience)`
This method sets the WildPokemon's `patience` value to `newPatience`.

`public void setSpeed(int newSpeed)`
This method sets the WildPokemon's `speed` to `newSpeed`.

`public void incrementTimeEscapedFromBall()`
This method increments the `timesEscapedFromBall` by one.

Three other methods:

`public void appear()`
This method will print a message.
`You encounter a wild [pokemonName]!`
The Pokemon name is from the parent class Pokemon. Then, use the parent class method `speak()` to make your pokemon speak.
For example:

```
You encounter a wild Pikachu!
Pikachu: pikapika!
```

`public boolean disappear()`
This method checks if the WildPokemon is going to disappear. If the wild pokemon's `patience` is less than or equal to 0, or if the WildPokemon has escaped from a ball for more than (or strictly greater than) 3 times, then this method will `return true`. Otherwise, `return false`. For the case when it is true, print out a statement: `[PokemonName] disappears...`

For example:

```
Pikachu disappears...
```

`public boolean isCaught(Berry berry, Pokeball pokeball)`
This method should simply `return true`.

Override `toString` method:
`public String toString()`
Override the `toString` method so it returns a string representation of the object. Return a string that represents the `name` of the Pokemon, specify that it is a `"WildPokemon"`, the `type` of the pokemon, `patience`, `speed`, and `timeEscapedFromBall` values. For example:

```
Pikachu, WildPokemon
type: electric
patience: 60
speed: 25
timeEscapedFromBall: 0
```

Test `WildPokemon.java` in `PA8Tester.java`

1. Create 2 WildPokemon objects with different `names`, `sounds`, `types`, `patience`, and `speed` (They should be different pokemons from Part 3 when you were testing PalPokemons).
2. Print the `name`, `sound`, `type`, `patience`, `speed`, and `timesEscapedFromBall` of each WildPokemon on a separate line using getter methods.
3. Also, make a function call to `appear()` and `disappear()` (we are going to add tests for `toString` method in `Part 5`)

Below are five kinds of pokemons you can choose:

Pokemon Name	Sound	Type	Patience	Speed
Pikachu	pikapika	electric	60	25
Bulbasaur	bulb	grass	50	5
Charmander	char	fire	50	20
Squirtle	squir	water	50	10
Mew	mew	psychic	15	50

Part 5: `Backpack.java` and `Pokedex.java` (Implementing and testing) (Pair programming allowed) (1.5 point)

Implement `Backpack.java`.

This class has one *private* member variable:

- **`ArrayList<Item> myItems`**: List of items to store pokeballs and berries.

This class should contain the following *public* methods. **You must name your methods *exactly* as specified below:**

One constructors:
`public Backpack ()`
This constructor should initialize a resizable-array ArrayLists which is the member variables of this Backpack class.

Other methods:
`public void add(Item item)`
This method takes a Item object and adds it to `myItems` ArrayList

`public void display()`
This method prints out all the items in the backpack. Start printing out what is in `myItems` ArrayList by stating `"Items in the backpack:"`. Make use of the `toString()` method implemented in previous parts. If `myItems` is empty, print `"None"`.

Below are 2 examples of the output:

-
```
Items in the backpack:

None
```
-
```
Items in the backpack:

Super Ball
performance: 10

Ultra Ball
performance: 30

Razz Berry
patienceIncrement: 10
speedDecrement: 0
```

Implement `Pokedex.java`.

This class has one *private* member variables:

- **`ArrayList<Pokemon> myPokedex`**: List of Pokemons to store wild pokemons and pal pokemons.

This class should contain the following *public* methods. You must name your methods *exactly* as specified below:

<u>One constructors:</u>
`public Pokedex ()`
This constructor should initialize a resizable-array ArrayLists which is the member variables of this Pokedex class.

<u>Other methods:</u>
`public void add(Pokemon pokemon)`
This method takes a pokemon object and adds it to `myPokedex` ArrayList.

`public void display()`
This method prints out all the Pokemons in the Pokedex. Start printing out what is in `myPokedex` ArrayList by stating `"Pokemons in the pokedex:"`. Make use of the `toString()` method implemented in previous parts. If `myPokedex` is empty, print `"None"`.

Below are 2 examples of the output:

- ```
  Pokemons in the pokedex:

  None
  ```
- ```
  Pokemons in the pokedex:

  Squirtle, PalPokemon
  pokeballName: superball
  type: water

  Mew, PalPokemon
  pokeballName: ultraball
  type: psychic
  ```

Test `Backpack.java` and `Pokedex.java` in `PA8Tester.java`

1. Display the empty backpack by calling `display()` method
2. Add 2 pokeballs that were created in Part 1 to the backpack and display the backpack
3. Add 2 berries that were created in Part 2 to the backpack and display the backpack
4. Display the empty pokedex by calling `display()` method
5. Add 2 pal pokemons that were created in part 3 to the Pokedex and display the Pokedex
6. Add 2 wild pokemons that were created in part 3 to the Pokedex and display the Pokedex

Part 6: Questions about subclasses and polymorphism (0.5 points) (Individual)

Answer a few short questions about subclasses and polymorphism as it applies to this assignment in [this Gradescope assignment].

Part 7: Recursion Warmup (3 points) (Pair programming allowed)

Write the following methods in a class named RecursionWarmup in a file named `RecursionWarmup.java`.

Recursion Question #1: Print in Binary [1.5 points]
Write the following recursive method:

```
public static String binaryString(int n)
```

This method should return the binary representation of the non-negative decimal number `n` to as a String. For example, calling ...

`binaryString(6)` returns `"110"`
`binaryString(12)` returns `"1100"`
`binaryString(17)` returns `"10001"`

You can assume `n` will be non-negative (greater than or equal to 0).

Hints:

- The *rightmost* (least significant) digit of the binary representation of a decimal number can be obtained by mod'ing that number by 2. *E.g.* `6%2 == 0`; `17%2 == 1`.
- The remaining digits to the left are the result of converting `n/2` to binary. *E.g.* `"11"` is 6/2 (3) in binary, and `"1000"` is 17/2 (8) in binary.
- A non-negative number less than 2 (that is, 1 or 0) is its own binary representation.

Test your method: In a `main` method in `RecursionWarmup.java`, add tests to test this method. You will not be graded on these tests, but it is up to you to make sure that your method works correctly. The autograder tests will not be exposed to you.

Recursion Question #2: Subset Sum [1.5 points]

Write the following recursive method:

```
public static boolean isSubSetSum(ArrayList<Integer> set, int targetNumber)
```

This method takes an ArrayList of integers, and a targetNumber and determines if there is a subset of those integers that sum to that targetNumber.

For example, given the numbers `{3, 7, 1, 8, -3}` and the target sum 4, the result is true because the subset `{3, 1}` sums to 4. On the other hand, if the target sum were 2, then the result is `false` since there is no subset that sums to 2.

Hints:

- If the set is empty, then it clearly has no elements that can add to the `targetNumber`.
- The subset of elements that are used in the sum will either contain the first number in the set, or it will not. Either way, once you have chosen to use the number or throw it away, it will need to be removed from the set you pass into your recursive call. You will need to have two recursive cases here: one where the first number *is* in the subset, and one where the first number is not in the subset. Whether you use the first element or not will affect what you pass in for targetNumber in your recursive call. Then, after you have done both recursive calls, consider how to combine them to give you your final answer. (It might make things faster to do only one, and only do the second if you need to).

Test your method: In a `main` method in `RecursionWarmup.java`, add tests to test this method. You will not be graded on these tests, but it is up to you to make sure that your method works correctly. The autograder test will not be exposed to you.

Star Point Options:

The star point this week is completely open-ended, as usual. Feel free to get creative with extensions. You'll have additional opportunities for more extensions next week.

Submission Instructions

Very important! Please follow the instructions below carefully and make the exact submission format. This is important since we will use scripts to grade so if you don't follow the same submission format you probably will receive a zero.

- Submit your code from Parts 1 to 5 (`Pokeball.java`, `Berry.java`, `PalPokemon.java`, `WildPokemon.java`, `Backpack.java`, `Pokedex.java`, and `PA8Tester.java`) on Gradescope using the "PA8 - Code (Parts 1 - 5)" submission
- Enter the reply to the questions about subclasses and polymorphism from Part 6 on Gradescope using the "PA8 - Part6" assignment
- Submit your code to the recursion practice problems from Part 7 on Gradescope using the "PA8 - Recursion Warmup (Part 7)" submission

Note: this programming assignment is out of 12 points, however, it's still worth the same as all the other assignments.