TP-Pokemon

Consignes

Vous devez répondre aux questions puis développer le projet ci-après, en respecant le diagramme de classe fourni.

NB : Le diagramme ne fait état que de la logique métier. Il vous appartient d'y intégrer des fonctions (et/ou propriétés) supplémentaires si nécessaire, afin de permettre la réalisation des opérations demandées.

Copiez-collez ce code sur le site 'plantext.com'

```
@startuml

skinparam classAttributeIconSize 0

interface Pokemon {
  nom!: str
  pv: int
  pvMax: num
  type: str
  captif: bool
  horsJeu: bool
  attaquer()
  subirAttaque()
  soigner()
  renommer()
}

interface Type {
  degats: num

  attaquerFeu(): num
  attaquerEau():num
  attaquerPlante(): num
}

abstract class TypeFeu {
}

abstract class  TypeEau {
}

abstract class  TypePlante {
}

class Salameche {
}

class Bulbizarre {
```

```
}

class Carapuce {
}
note bottom: constructor : pvMax, degats

class Pokeball {
  contient!: Pokemon
  proprietaire!: Dresseur
  +getContenu()
  +affecterPokemon()
}

class Dresseur {
  -nom: str
  +ajouterPokeballs( int )
  +capturer( Pokemon )
  +getPokemons()
}

TypeFeu -up-|> Type
TypeEau -up-|> Type
TypePlante -up-|> Type

Salameche -up-|> TypeFeu
Carapuce -up-|> TypeEau
Bulbizarre -up-|> TypePlante

Salameche --|> Pokemon
Carapuce --|> Pokemon
Bulbizarre --|> Pokemon

Dresseur "1" -- "0..6" Pokeball
Pokeball "1" -- "0..1" Pokemon

@enduml
```

Questions

Répondez aux questions suivantes (sur 10 points) : NB : Il y a plus de points que nécessaire. Les points seront 'bonus'

1. Le diagramme respecte-t-il le principe de responsabilité unique ? (1 point)
2. Pourquoi ? (1 point)
3. Le diagramme respecte-t-il le principe Open/Closed ? (1 point)
4. Pourquoi ? (1 point)
5. Le diagramme respecte-t-il le principe de substitution de Liskov ? (1 point)
6. Pourquoi ? (1 point)
7. Le diagramme respecte-t-il le principe de ségrégation des interfaces ? (1 point)
8. Pourquoi ? (1 point)
9. Le diagramme respecte-t-il le principe d'inversion des dépendences ? (1 point)

10. Pourquoi ? (1 point)
11. Proposez un diagramme alternatif (2 point)
12. Déposez votre projet sur github et précisez le lien (1 point)

Concernant la correction du code

- La correction des questions vous sera envoyée
- Le formateur utilisera son propre fichier 'index.ts' pour tester les différents aspects de votre code. Pour avoir tous les points :
    - Respectez la nomenclature des classes, instances, paramètres et fonctions du diagramme !
    - Ne joignez pas de fichier 'index.ts' à votre projet.
    - Respectez la nomenclature 'nom-prenom_Pokemon' pour le nom du projet.
- Vous devez impérativement respecter l'arborescence suivante :

```
_Pokemon
|
|_Classes
|   |_Bulbizarre.ts
|   |_Carapuce.ts
|   |_Dresseur.ts
|   |_Pokeball.ts
|   |_Salameche.ts
|   |_TypeEau.ts
|   |_TypeFeu.ts
|   |_TypePlante.ts
|
|_Interfaces
|   |_Pokemon.ts
|   |_Type.ts
```

> La correction étant faite à l'aide d'un script, **votre code ne pourra pas être vérifié si cette arborescence n'est pas respectée** ! Ce serait dommage de perdre des point pour ça !

Aides du Poussin

Lexique de l'UML :

- I : Interface,
- A : Classe abstraite,
- C : Classe
- Prêtez une attention particulière aux héritages et cardinalités.
    - Qu'est-ce que ces dernières impliquent ?
- Certaines fonctions vont en appeler d'autres, d'un objet à l'autre !

Interface Type

Cette interface ne sert qu'à donner les méthodes aux différentes classes abstraites

```
degatsFeu(): number;
degatsEau(): number;
degatsPlante(): number;
```

Pour chaque classes abstraite qui descend de Type

N'oubliez pas que votre fonction doit retourner les dégâts uniquement Si la classe est abstraite, la fonction ne l'est pas !

Pour chaque classe Pokemon

```
// Constructeur
constructor(pvMax: number, degats: number) {
    //...
}
// Fonction attaquer, qui devra récupérer le type du pokemon cible, et utiliser la
méthode appropriée pour calculer les dégâts
attaquer(cible: Pokemon): number
// Permet d'éditer les PV du pokemon
subirAttaque(degats: number): void
// Si le pokemon n'a plus de PV, il ne peut plus attaquer
checkVivant(): void
// Permet de restaurer les PV d'un pokemon
soigner(): void
// Permet de donner un surnom au pokemon
renommer(nom: string): void
```

Interface Pokemon

Reprend les fonctions qui doivent être présente dans les classes pokemon C'est ici que vous pouvez taper la doc de vos fonctions !

Classe Dresseur

```
// Propriétés :
private nom: string;
public pokeballs!: Pokeball[];
// Constructeur
constructor(nom: string)
// Getter :
get nbPokeballs()
// Methodes :
```

```
// On ajoute 1 ou plus pokeballs, dans un maximum de 6 !
ajouterPokeballs(nombre: number): void

// Vérifie si le dresseur a encore des pokeballs dispo
private getPokeballs(): boolean
// affecte un pojemon à une pokeball, et précise son maitre au passage
private fillEmptyPokeball(cible: Pokemon, dresseur: Dresseur): void
// Permet de vérifier si un pokemon n'a pas déjà été capturé,
// Appelle 'fillEmptyPokeball'
capturer(cible: Pokemon): void
// Affiche les pokemons contenus dans chage pokeball
getPokemons()
```