Examen C#

Cet examen comporte tous ce que vous avez appris durant l'entière de la piscine C#
Vous disposez de 2h avant de rendre votre copie

Durant cet examen, nous allons travailler sur un début de jeu vidéo, plus précisément nous allons travailler sur la création de personnage et de leur classe/spécialité.

1) Character

Nous allons commencer par définir ce qu'est un personnage.
Crée d'abord un fichier nommé Character.cs, avec pour squelette :

```
using System;

namespace CSharpDiscovery.Examen
{
    public abstract class Character
    {
        //TODO
    }
}
```

Vous devrez définir les variables suivantes, elles seront publiques :

- Nom (String) : Le nom du personnage
- Hp (Float) : Nombre de points de vie actuelle
- HpMax (Float) : Nombre de points de vie maximum
- CreationDate (DateTime) : Date de création du personnage, initialisé a aujourd'hui, statique

Vous allez ensuite ajouter 2 constructeurs
Le premier ne prend pas de paramètre et affecte les valeurs de la sorte :

- Nom : « Npc»
- Hp(Float) : 100
- HpMax (Float) : 100

Le second constructeur va prendre 2 paramètres et définira le Nom, HpMax.
Le Hp sera définie avec la même valeur que HpMax.

Enfin vous définirez différente fonction utile à nos personnages

TakeDamage	Prend un Int en paramètre Ne renvoie rien	Change les points de vie (Hp) d'un personnage, ne peut pas descendre en dessous de 0
GetCreationDate	Ne prend pas de paramètre Renvoie un String	Renvoie la valeur de CreationDate au format « Jour/Mois Heure :Minute »
ToString	Redéfinir ToString Ne prend pas de paramètre Renvoie un String	Permet de renvoyer une chaine de caractère sous la forme « Nom : Hp/HpMax»
Special	Ne prend pas de paramètre Ne renvoie rien Définie de façon abstrait	Notre première compétence, il est de type abstrait pour le définir dans les autres fichiers
CibledSpecial	Prend un Character en paramètre Ne renvoie rien Définie de façon abstrait	Notre seconde compétence, il est de type abstrait pour le définir dans les autres fichiers Celle-ci prend en paramètre un personnage cible

2) Spécialisation : Warrior.cs, Cleric.cs

Maintenant que nous avons un personnage nous pouvons le spécialiser
Crée deux fichiers : Warrior.cs et Cleric.cs

```
using System;

namespace CSharpDiscovery.Examen
{
    public class Warrior // De meme avec Cleric
    {
        //TODO
    }
}
```

Ces 2 classe hérite de Character, vous utiliserez le mot clé base() pour les constructeur , l'un des constructeurs ne prend pas de paramètre , l'autre récupère le valeur de Bravery/Mana (en fonction de la classe) puis le Nom et les HpMax

Tache	Warrior	Cleric
Ajout de variable	Rajouter la variable public « Bravery » qui est un boolean , initialisé a faux	Rajouter la variable public « Mana » qui est un double initialisé a 100
Redéfinir Special	Permet, si les points de vie (Hp) sont en dessous de 30 de changer « Bravery » a vrai Sinon ne fait rien	Permet d'augmenter le « Mana » de 10, le « Mana » Le Mana ne peut pas dépasser 100
Redéfinir CibledSpecial	Baisse de 25 les points de vies (Hp) de la cible Si « Bravery » est vrai alors ajoute 15 supplémentaire Si les points de vie de la cible devraient devenir négatif ils deviennent 0	Augmente de 15 les points de vies (Hp) de la cible Les points de vie (Hp) ne peuvent pas dépasser les HpMax de la cible
Redéfinir ToString	ToString doit renvoyer une chaine sous la forme « Nom : Hp/HpMax | Classe : Guerrier | Bravoure : Bravery»	ToString doit renvoyer une chaine sous la forme « Nom : Hp/HpMax | Classe : Clerc | Mana : Mana»

3) Interface : Archetype

Désormais nous avons des personnage avec des classes, nous pouvons catégorisé les classe par archetype, chaque archétype partage les même compétence entre eux
Crée un fichier nommé Archetype.cs dans le quelle vous définirez les interfaces IHealer et ITank
L'interface IHealer doit contenir :

- Un Integer HealPower
- Une méthode DoubleHeal qui prend 2 Character et ne revoie rien
- Une méthode GetHeal qui ne prend pas de paramètre et renvoie un Integer

Vous intégrée cette interface a votre fichier Cleric.cs

- HealPower sera initialiser dans les constructeurs a 15
- Vous changerez la valeur dans CibledSpecial par celle de HealPower
- DoubleHeal fonctionne comme CibledSpecial mais prend 2 Character a la place d'un seul, le soin sera divisé par 2 entre les cibles

L'interface ITank doit contenir :

- Un Integer AttacklPower
- Une méthode BigAttack qui prend 1 Character et ne revoie rien
- Une méthode GetPower qui ne prend pas de paramètre et renvoie un Integer

Vous intégrée cette interface a votre fichier Warrior.cs

- AttacklPower sera initialiser dans les constructeurs a 25
- Vous changerez la valeur dans CibledSpecial par celle de AttackPower
- BigAttack est une fonction qui retire 10 points de vie (Hp) au personnage qui l'utilise puis retire AttackPower * 2 points de vie (Hp) au Character pris en paramètre

Vous crée ensuite un fichier nommé Paladin.cs qui hérite de Character et inclue les interfaces IHeal et ITank vous utiliserez le mot clé base() pour les constructeur , l'un des constructeurs ne prend pas de paramètre , l'autre récupère le valeur de Buff (en fonction de la classe) puis le Nom et les HpMax.

Tache	Warrior
Ajout de variable	Rajouter la variable public « Buff » qui est un int , initialisé a 0
Redéfinir Special	Augmente de la valeur HealPower + « Buff » ces points de vies (Hp) Les points de vie (Hp) ne peuvent pas dépasser les HpMax
Redéfinir CibledSpecial	Baisse de AttackPower + « Buff » les points de vies (Hp) de la cible Augmente la valeur de « Buff » par 3 , « Buff » ne peut pas dépasser 15 Si les points de vie de la cible devrais devenir négatif ils deviennent 0
Redéfinir ToString	ToString doit renvoyer une chaine sous la forme « Nom : Hp/HpMax | Classe : Paladin | Buff : Buff»