Programmation objet

Héritage

Les héros

- Gestion de types de héros ayant des caractéristiques différentes

Classe Heros

- Informations d'un héros: nom, points de vie, puissance d'attaque, bouclier de défense

- Opérations pouvant être réalisées pour héros:
  - Affichage des informations du héros
  - Retirer des points de vie
  - Connaitre son nom, sa puissance d'attaque, son bouclier défense, son état

Les différents héros

- Tous les héros ont les caractéristiques de la classe Heros (enlever la puissance)
- On aura deux sous-types de Heros :
  - Héros de Terre
  - Héros de Feu

Héros de terre

- Ces héros sont caractérisés par un poids (en kg) et un degré d'agilité.
- La puissance de ces héros sera calculée grâce à la formule suivant : *puissance =poids/50 * degré d'agilité*

Héros de feu

- Ces héros sont caractérisés une taille (en centimètres) et une puissance de feu.

- La puissance des ces héros sera calculée grâce à la formule suivante : *puissance = taille/100 * puissance de feu*

Conception des classes

- Mise en œuvre de l'héritage pour éviter de répéter le code permettant de gérer les informations et comportements communs des héros.

Exercice

- Représenter le diagramme de classes de la structure des Héros

Héritage

- La classe Heros est appelée classe de base ou classe mère ou super classe

- Les classes HerosTerre, HerosFeu héritent de la classe Héros
- Les classes HerosTerre, HerosFeu sont appelées classe dérivée, classe spécialisée, classe fille ou sous-classe

Principe de l'héritage

- Les classes HerosTerre, HerosFeu héritent de la classe Heros
  - Les attributs protected
  - Les méthodes public et protected

Principe de l'héritage

- Un objet HerosTerre, HerosFeu pourra être utilisé dans le programme comme un objet Heros
  - dans un tableau de Heros par exemple
- Un objet Heros ne pourra pas être utilisé comme un objet HerosTerre, HerosFeu car il n'a pas toutes les propriétés de ceux-ci.

Définition de l'héritage

- On définit qu'une classe dérive d'une autre classe dans la déclaration de la sous-classe

Public class NomClasseDérivée extends **NomClasseBase**
{

};

Classe de Base

- Les attributs définis avec une portée private ne pourront pas être utilisés dans les méthodes des classes dérivées.
- Dans la classe de base, les attributs qui pourront être accessible par les classes dérivées devront être définis avec la portée protected

Exercice

- Modifier la classe Heros pour définir la portée protected
- Créer la classe HerosTerre qui hérite de Heros
- Ajouter les attributs spécifiques de HerosTerre dans la classe

Constructeur dans la classe dérivée

- La classe dérivée ne peut pas initialiser directement les attributs de la classe de base
- Dans le constructeur de la classe dérivée, on peut appeler un constructeur de la classe de base

Exemple

- Le constructeur de la classe HerosTerre doit initialiser
  - les attributs hérités: nom, points de vie, bouclier de défense
  - Ses attributs propres : poids agilité
- On va utiliser le constructeur de la classe Heros qui initialise les attributs nom, points de vie, bouclier de défense
- Pour faire référence à la classe de base on utilise le mot clé super
- Exemple : Appel du constructeur vide de la classe de base : super();

Exercice

- Créer le constructeur de la classe HerosTerre qui permet d'initialiser tous les attributs de la classe (un paramètre par attribut à initialiser).

- Créer le constructeur de la classe HerosTerre qui prend en paramètre un nom de héros et initialise ses propriétés avec des valeurs par défaut (poids : 150, agilité : 70)

Exercice :Méthode toString

- Créer la méthode toString dans la classe Heros pour retourner une description de l'objet sous forme de chaine de caractères

- Créer la méthode toString dans la classe HerosTerre en réutilisant la méthode de la classe Heros et ajoutant les informations spécifiques au HerosTerre

Exercice : méthode puissance

- Créer une méthode puissance() qui retourne la puissance du héros de Terre selon la formule suivante :
- *puissance =poids/50 * degré d'agilité*

Création des objets

- Au niveau de la création des objets, la relation d'héritage ne modifie pas la syntaxe
- Un objet de la classe dérivée peut être utilisé comme un objet de la classe de base
- Un objet de la classe de base ne peut pas être utilisé comme un objet de la classe dérivée.

Exercice

- Créer un héros et 2 héros de terre
- Afficher les informations de ces 3 objets et la puissance des héros

Méthode combat

- Pour un Héros Terre, le combat se déroule de la façon suivante:
  - Si la puissance du héros qui attaque est supérieure à la puissance du héros en défense, l'attaquant fait perdre 40 points au défenseur.
  - Si les deux puissances sont égales, aucun ne perd de points
  - Si la puissance du défenseur est supérieure à celle de l'attaquant, celui-ci perd 15 points.
- A la fin de la méthode combat, si un héros est mort, afficher un message indiquant le nom du héros gagnant le combat et le nom du héros tué.

Exercice

- Créer la classe HerosFeu avec
  - La relation d'héritage
  - Ses attributs spécifiques
  - le constructeur qui prend en paramètres des valeurs pour initialiser tous les attributs
  - Le constructeur par défaut avec le nom en paramètres, une taille de 110 cm et une puissance de feu de 60
  - La méthode toString
  - La méthode puissance()
- Tester la classe

HerosFeu - combat

- La méthode combat fonctionne de la façon suivante :
  - Si la puissance du héros qui attaque est supérieure à la puissance du héros en défense, l'attaquant fait perdre 50 points au défenseur.
  - Si les deux puissances sont égales et qu'il reste au moins 50 points de vie à l'attaquant, il fait perdre 25 points au défenseur.
  - Si la puissance du défenseur est supérieure à celle de l'attaquant et qu'il reste au moins 50 points de vie à l'attaquant, le défenseur perd 15 points. Si le nombre de points de vie de l'attaquant n'est pas suffisant, celui-ci perd 10 points.

- A la fin de la méthode combat, si un héros est mort, afficher un message indiquant le nom du héros gagnant le combat et le nom du héros tué.

Mise en œuvre du polymorphisme

- Principe : uniquement utilisable avec de l'héritage
- Utilisation de différentes méthodes (poly) selon l'objet précis qui l'appelle (morphisme)

Test des héros

- Créer plusieurs héros qui seront stockés dans un tableau de Héros

- Appeler la méthode toString() sur tous les héros du tableau
  - Résultat attendu : en fonction du type de Héros (terre, feu), les informations affichées sont différentes.

Test des héros

- Vérifier ce comportement avec la méthode puissance