

Programmation objet

Création d'une classe

Projet

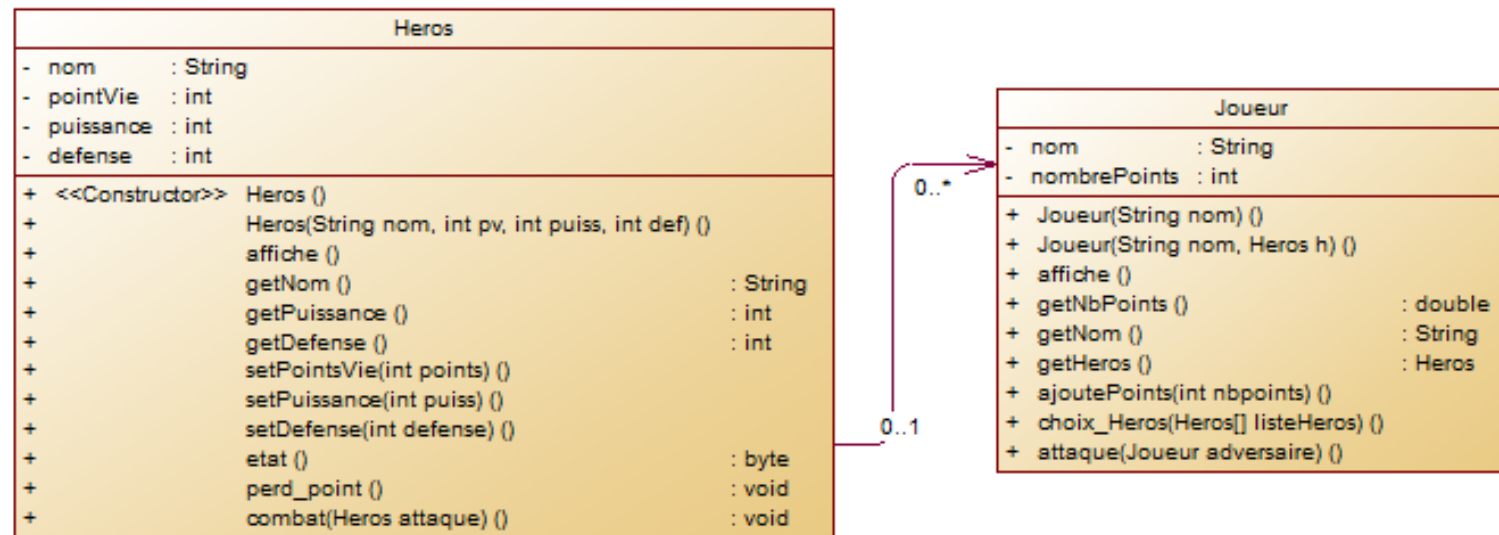
- Création d'un jeu basé sur des combats de héros



Classes du projet

- Classe Héros permettant de gérer différents héros et leurs combats
- Classe Joueur permettant de gérer les joueurs et leurs actions

Diagramme de classes





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



Classe Joueur

- Informations d'un joueur : nom, nombre de points, héros
- Opérations pouvant être réalisées pour un joueur:
 - Afficher le détail du joueur
 - Choisir un héros
 - Lancer une attaque
 - Connaitre son nom, son nombre de points, son héros

Création de classes

- Afin de permettre une bonne organisation des programmes, chaque classe est créée dans un fichier qui lui est propre.
- Le fichier porte le nom de la classe et l'extension du langage .java

Création de classe

- Dans NetBeans, on ajoute une classe dans la partie fichiers sources du projet, avec un clic droit : New → JavaClass

```
public class Heros {  
  
}
```


L'encapsulation

Encapsulation

- L'accès aux éléments d'une classe peut être restreint
 - À la classe
 - À la classe et ses sous-classes
 - Sans restriction

Encapsulation

- Sécurité des attributs de la classe
- Limiter l'utilisation des méthodes internes à la classe
- Le mot clé qui définit la portée précède chaque nouvel élément de la classe (attribut ou méthode)

Portée des éléments de la classe

- private : Utilisé pour les attributs et les méthodes internes de la classe
 - Les éléments de cette section sont accessibles uniquement par les méthodes définies dans la classe
- public : Utilisé pour la plupart des méthodes
 - Les éléments de cette section sont accessibles par tous les programmes

Portée des éléments de la classe

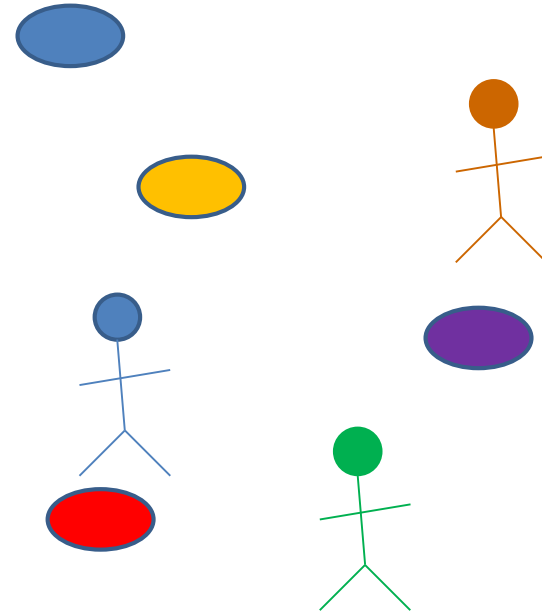
- protected : Utilisé par les attributs lorsque la classe peut être héritée
 - Les éléments de cette section sont accessibles uniquement par les méthodes définies dans la classe et dans les classes qui en héritent

Comportement de l'application

Programme

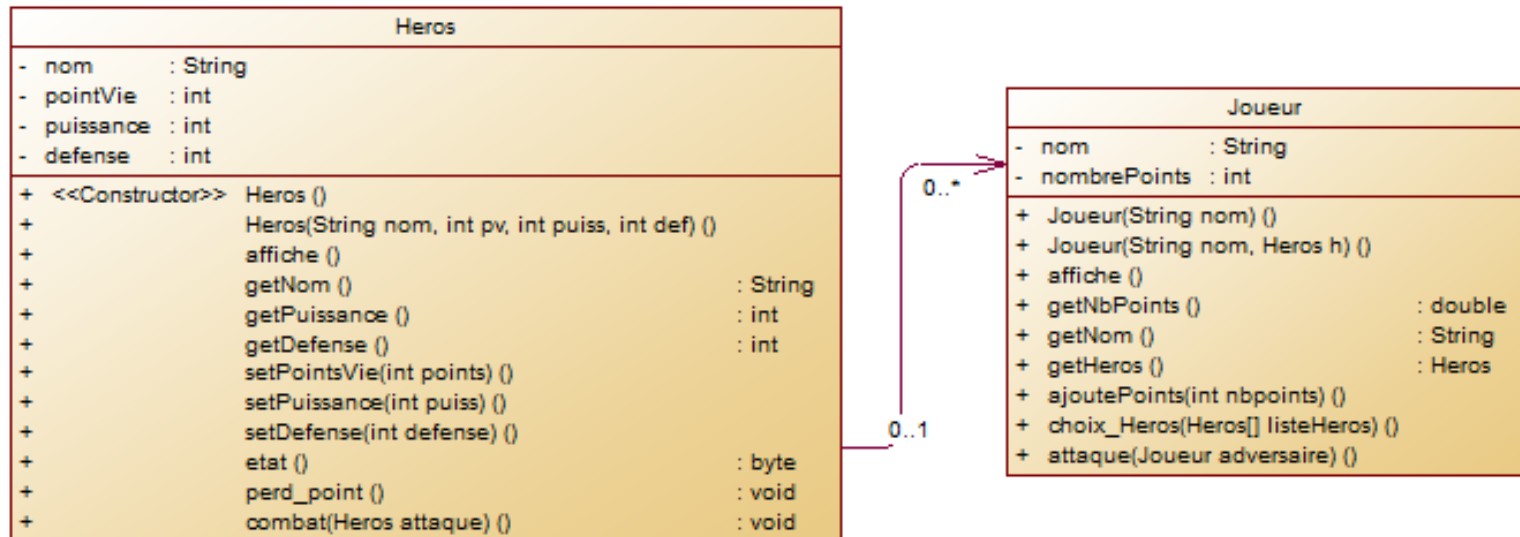
- Le programme manipule des objets
- Exemple : des joueurs et des héros

- En mémoire



Programme

- Des classes sont à disposition du programme



Programme

- Le programme peut utiliser uniquement les éléments de portée public de la classe

Heros		
+	<<Constructor>> Heros ()	
+	Heros(String nom, int pv, int puiss, int def) ()	
+	affiche ()	
+	getNom ()	: String
+	getPuissance ()	: int
+	getDefense ()	: int
+	setPointsVie(int points) ()	
+	setPuissance(int puiss) ()	
+	setDefense(int defense) ()	
+	etat ()	: byte
+	perd_point ()	: void
+	combat(Heros attaque) ()	: void

Création des objets

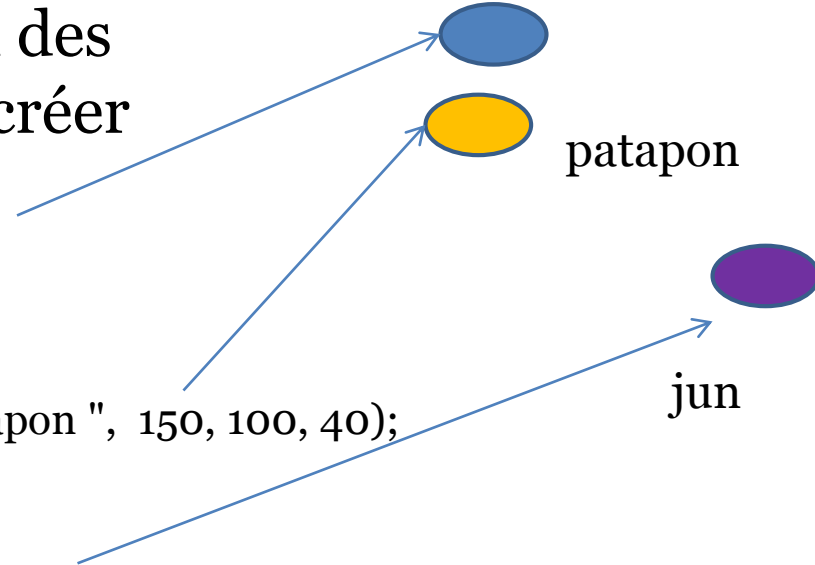
- Pour créer des objets, le programme doit utiliser les constructeurs en respectant les paramètres qui ont été définis



Création des objets

- On déclare une variable du type de la classe et on appelle un des constructeurs pour le créer
- `Heros gigatron = new Heros();`
- `Heros patapon = new Heros("patapon ", 150, 100, 40);`
- `Heros jun = new Heros("junon ", 90, 120, 80);`

- Mémoire



Création des objets

- Chaque objet de la classe possède la même structure
- Des informations cachées du programme

- jun


nom
pointsVie
attaque
defense

junon
90
120
80



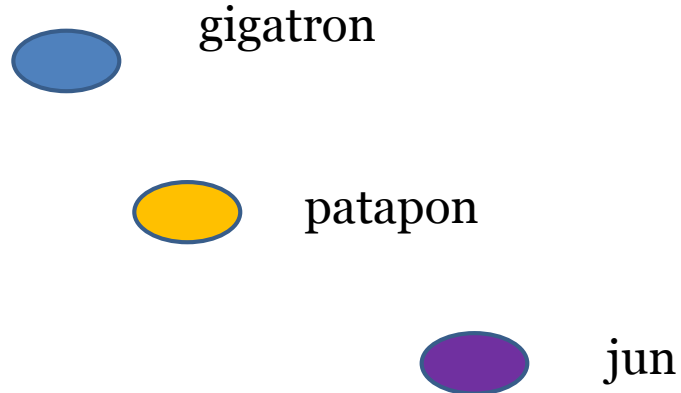
Création des objets • jun

- Chaque objet de la classe possède la même structure
- Des méthodes permettant au programme de réaliser des opérations sur l'objet



```
affiche ()  
getNom ()  
getPuissance ()  
getDefense ()  
setPointsVie(int points) ()  
setPuissance(int puiss) ()  
setDefense(int defense) ()  
etat ()  
perdre_point ()  
combat(Heros attaque) ()
```

Opérations sur les objets



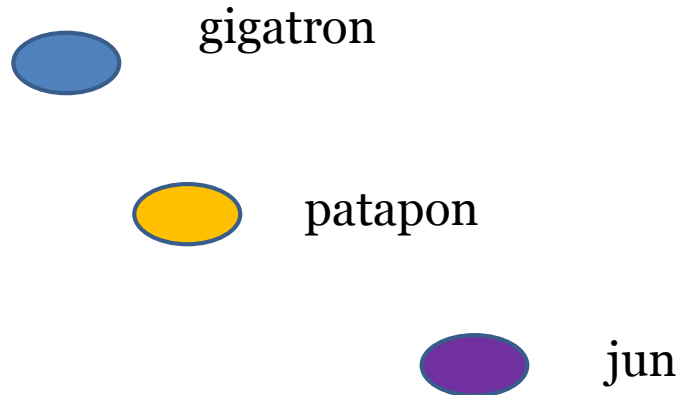
Opérations possibles pour chaque objet :

```
affiche ()  
getNom ()  
getPuissance ()  
getDefense ()  
setPointsVie(int points) ()  
setPuissance(int puiss) ()  
setDefense(int defense) ()  
etat ()  
perd_point ()  
combat(Heros attaque) ()
```

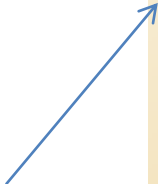
Programme

- On veut afficher les informations du héros junon
- Il faut utiliser l'objet représentant ce héros (à travers son nom de variable) et appeler la méthode d'affichage

Opérations sur les objets



Opérations possibles pour chaque objet :



```
affiche ()  
getNom ()  
getPuissance ()  
getDefense ()  
setPointsVie(int points) ()  
setPuissance(int puiss) ()  
setDefense(int defense) ()
```

```
jun.affiche();
```

```
Junon  
90 points de vie  
Puissance d'attaque : 120  
Défense : 80
```


Les attributs

Ajout d'attributs pour la classe

- Les attributs sont les variables permettant d'enregistrer les informations que l'on souhaite conserver pour chaque objet.
- Les attributs sont généralement de portée privée (private) afin de limiter les erreurs dues à des modifications de données non contrôlées

Ajout d'attributs pour la classe

- On définit pour chaque attribut sa portée, son type et son nom.
- Le mot clé final définit après la portée d'un attribut permet de déclarer une constante (valeur nom modifiable)

Attributs de la classe Heros

```
public class Heros {  
    private String nom;  
    private int pointsVie;  
    private int puissance;  
    private int defense;
```

Les constructeurs

Constructeurs et méthodes

- Les constructeurs permettent de créer un objet de la classe
- Les méthodes permettent de définir les opérations que l'on peut réaliser sur un objet de la classe

Création des constructeurs

- Un constructeur est une méthode permettant de créer un objet de la classe
- Une classe dispose obligatoirement d'au moins un constructeur
- Le constructeur peut permettre d'initialiser les valeurs des attributs de la classe



gigatron



patapon



jun

Création des constructeurs

- Un constructeur est créé dans la section public de la classe
- Un constructeur possède toujours le même nom que la classe
- Un constructeur peut avoir 0, 1 ou plusieurs paramètres

Constructeurs de la classe Heros

- Un constructeur sans argument qui initialise toutes les valeurs de chaîne à vides et les valeurs numériques à 0
- Un constructeur avec un paramètre pour chaque attribut de la classe qui initialise les valeurs des attributs avec les valeurs passées en paramètre.
- Un constructeur simplifié avec le nom du héros qui initialise les propriétés du héros avec les valeurs par défaut suivante :
 - Points de vie : 100
 - Attaque : 50
 - Défense: 30

Constructeur sans argument

```
public Heros()  
{  
    nom="";  
    pointsVie=0;  
    puissance=0;  
    defense=0;  
}
```

Constructeur avec 4 paramètres

```
public Heros(String n, int pv, int puis, int def)
{
    nom=n;
    pointsVie=pv;
    puissance=puis;
    defense=def;
}
```

Constructeur par défaut

```
public Heros(String n)
{
    nom=n;
    pointsVie=100;
    puissance=50;
    defense=30;
}
```

Les méthodes

Création des méthodes

- Les méthodes permettent de définir les opérations qu'il sera possible de réaliser avec un objet de la classe une fois celui-ci créé
- Une méthode est une fonction ou une procédure qui peut avoir 0, 1 ou plusieurs paramètres
- Une méthode peut manipuler les attributs de la classe

Ecriture d'une méthode

- La méthode s'exécute à partir d'un objet
- Considérer que les attributs ont une valeur à laquelle les méthodes ont accès



gigatron



patapon



jun

Méthodes de la classe Héros

- Méthode affiche, pas de paramètre

Permet d'afficher les informations de l'objet qui appelle la méthode selon le format suivant

[NomHéros]

[Point de vie] points de vie

Puissance d'attaque: [attaque]

Bouclier de défense: [defense]

Méthode affiche

```
public void affiche()  
{  
    System.out.println(nom);  
    System.out.println(pointsVie + "points de vie");  
    System.out.println("Puissance d'attaque " + puissance);  
    System.out.println("Bouclier de défense " + defense);  
}
```

Méthodes de la classe Heros

- Méthode etat, pas de paramètre

Retourne un booléen indiquant si le héros est toujours vivant à partir du nombre de points de vie restant

Méthode etat

```
public boolean etat()  
{  
    if (pointsVie<=0)  
        return false;  
    return true;  
}
```

```
public boolean etat()  
{  
    return pointsVie<=0;  
}
```

Méthodes de la classe Heros

- Méthode `perd_point`, un entier en paramètre
Retire au héros le nombre de points passés en paramètre.

Méthode perd_point

```
public void perd_point(int points)
{
    pointsVie -=points;
}
```

Les accessoires

Les Accesseurs

- Méthodes particulières et standardisées permettant de donner un accès aux attributs d'une classe
 - En lecture (get)
 - En écriture (set)

Création d'un accesseur en lecture get

- Par convention, la méthode s'appelle `getNomAttribut`
- Sans paramètre
- Retourne l'information contenue dans l'attribut de l'objet

Exercice

- Ajouter une méthode getNom dans la classe Heros.
- Cette méthode permet d'obtenir l'information contenue dans l'attribut nom de la classe

Création d'un mutateur en écriture set

- Par convention, la méthode s'appelle `setNomAttribut`
- 1 paramètre du type de l'attribut
- Ne retourne rien
- Modifie la valeur de l'attribut avec la valeur donnée en paramètre

Accesseur set

- Remarque : Dans certains cas un accesseur peut modifier plusieurs attributs de la classe simultanément (lorsque les valeurs des attributs sont liées)

Exercice

- Ajouter une méthode setPuissance dans la classe Heros.
- Cette méthode permet de modifier l'information contenue dans l'attribut puissance de la classe

Exercice

- Ajouter les accesseurs et mutateurs du diagramme de classe dans la classe Heros

Le pointeur this

Le pointeur this

- A l'intérieur d'une classe d'objet, un pointeur sur l'objet courant est automatiquement créé : le pointeur `this`
- Le pointeur `this` est accessible dans toutes les méthodes de la classe et permet de représenter l'objet ayant appelé la méthode.

Méthode combat

- La méthode combat permet de gérer l'attaque de l'objet Héros qui appelle la méthode vers l'objet Héros passé en paramètre

Méthode combat

- `jun.combat(patapon)`

```
Public void combat(Heros attaque)  
{
```



Représenté par `this`

```
    //vérification de l'état de l'attaquant  
    this.etat();  
    //vérification de l'état de l'attaqué  
    attaque.etat();
```

```
}
```

Exercice

- Compléter la méthode combat pour lancer l'attaque uniquement si l'attaquant est en vie
- Le héros attaqué perd un nombre de points de vie équivalent à la puissance de l'attaque reçue moins la valeur de son bouclier.
- si le bouclier est supérieur ou égale à la puissance de l'attaquant, le défenseur ne perd pas de points
- A l'issue de l'attaque, si le héros attaqué est mort, afficher un message de victoire.

Test de la classe

Test de la classe

- Afin de vérifier le fonctionnement de la classe, nous devons créer des objets de la classe et utiliser chaque méthode créée
- Test de la classe dans la classe main

Création d'un objet

- Déclaration d'un objet (utilise le constructeur vide)
 - `TypeObjet nomObjet;`
- Création d'un objet avec un constructeur
 - `TypeObjet nomObjet = new TypeObjet(param1, param2...);`

Appel d'une méthode

- `nomObjet.nomMethode(param1,param2...);`

Jeu d'essai

- Créer un objet héros à partir du constructeur sans argument
- Créer un objet héros en initialisant chacun des paramètres
- Créer un objet héros à partir du constructeur par défaut
- Tester chacune des méthodes à partir de ces 3 objets

Réalisation des tests

	Objet utilisé	Résultat du test
affiche		
estVivant		
Perd_point		
getNom		
setPuissance		
combat		

Le destructeur

Destruction des objets

- Gérée par le Garbage Collector ou Ramasse Miettes
- Objectif : optimiser la gestion de la mémoire (RAM)

Destruction des objets

- Ajouter la méthode finalize dans la classe
- Méthode appelée automatiquement lors de la destruction de l'objet

Exemple :

```
public void finalize()  
{  
    System.out.println("Objet nettoyé de la mémoire");  
}
```