# Projet LU2IN002 - 2020-2021

Numéro du groupe de TD/TME : Groupe 7

| | | |
|---|---|---|
| *Nom :* Alliot | *Nom :* Aouchiche | *Nom :* Babaali |
| *Prénom :* Léane | *Prénom :* Juba | *Prénom :* Mohamed |
| *N° étudiant :* 28602178 | *N° étudiant :* 28610319 | *N° étudiant :* 28601465 |

*Thème choisi (en 2 lignes max.)*

Tournoi où s'affronteront les meilleurs représentants (ayant déjà existé ou légende) de chaque art martial. Le vainqueur pronera que son art martial est le meilleur de tous.

*Description des classes et de leur rôle dans le programme (2 lignes max par classe)*

**Personne :** classe dont hériteront tous les combattants, spectateurs et l'arbitre.

**Arbitre :** classe qui hérite de Personne et qui contient toutes les actions qui concernent l'arbitre.

**Spectateur :** classe qui hérite de Personne et qui contient la création des spectateurs par copie pour qu'ils puissent assister aux différents combats.

**Fighter :** classe qui hérite de Personne et qui permet de différencier les combattants des autres personnes

**Legende :** classe qui hérite de Fighter et qui désigne un combattant légendaire ou fictif avec une maîtrise parfaite de son art martial. Ils sont en effet plus forts que les combattants Moderne.(plus grosse stats de combat)

**Hercule :** classe qui hérite de Legende et qui contient toutes les actions du combattant Hercule qui représente l'art martial → la Lutte greco-romaine.

**Madara_Uchiwa :** classe qui hérite de Legende et qui contient toutes les actions du combattant Madara Uchiwa (Naruto Shippuden) qui représente l'art martial → le Ninjustu.

**Moderne :** classe qui hérite de Fighter et qui désigne un combattant humain réel connu pour sa maîtrise parfaite de son art martial.

**Bruce_Lee :** classe qui hérite de Moderne et qui contient toutes les actions du combattant Bruce Lee qui représente l'art martial → le Kung Fu.

**Chuck_Norris :** classe qui hérite de Moderne et qui contient toutes les actions du combattant Chuck Norris qui représente l'art martial → le Karaté.

**Mike_Tyson :** classe qui hérite de Moderne et qui contient toutes les actions du combattant Mike Tyson qui représente l'art martial → la Boxe.

**Hakuho_Sho :** classe qui hérite de Moderne et qui contient toutes les actions du combattant Hakuho Sho qui représente l'art martial → le Sumo.

**Teddy_Riner :** classe qui hérite de Moderne et qui contient toutes les actions du combattant Teddy Riner qui représente l'art martial → le Judo.

**Conor_McGregor :** classe qui hérite de Moderne et qui contient toutes les actions du combattant Conor McGregor qui représente l'art martial → le MMA.

**Technique :** interface qui permet aux combattants légendaires de maîtriser une technique que les combattants Moderne ne peuvent pas reproduire.

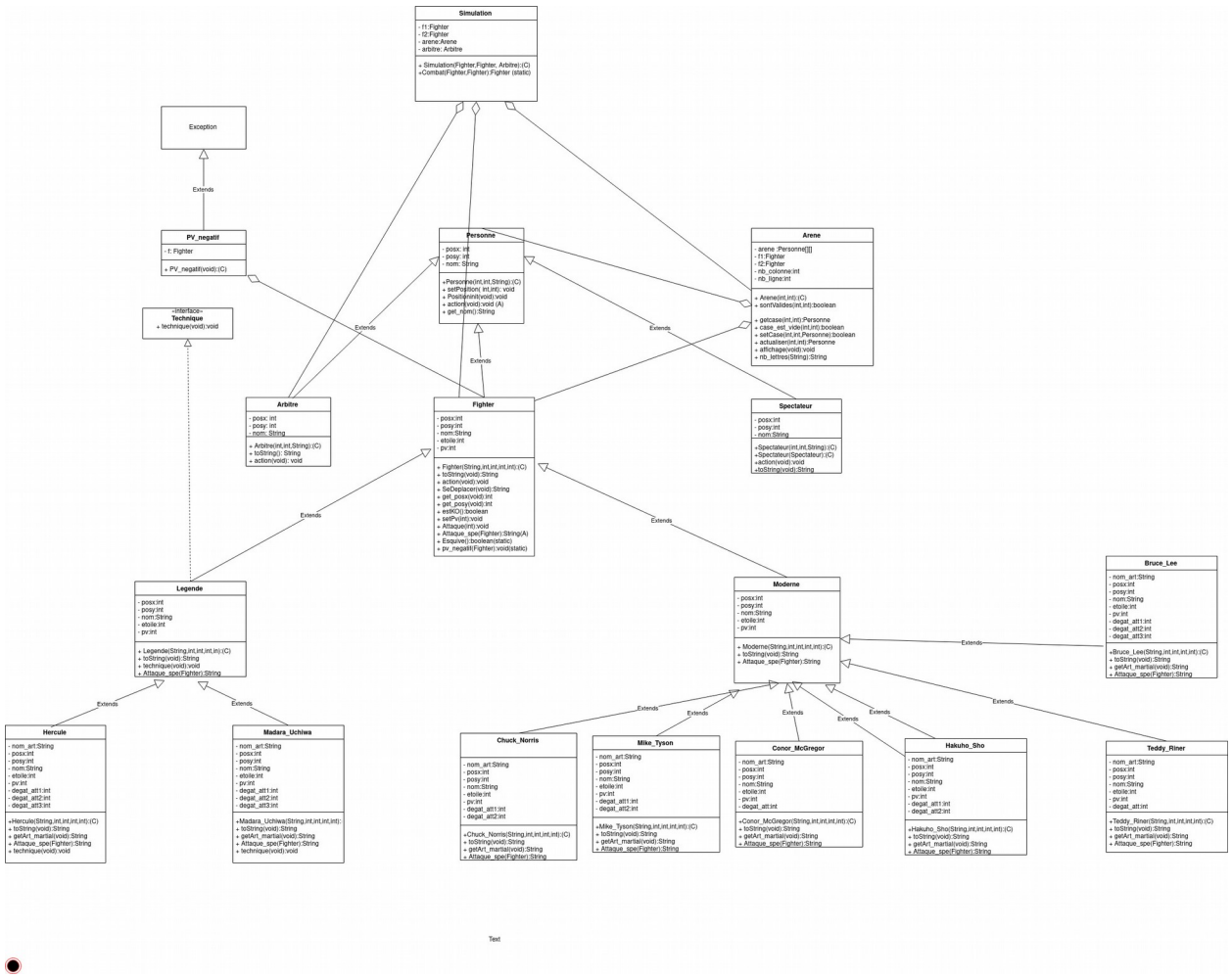
**PV_negatif :** classe Exception qui permet aux combattants d'avoir au minimum 0 point de vie (pv) et non un nombre négatif.( 0 pv est l'équivalent d'un K.O)

**Arene :** classe qui met en place un terrain où les specateurs pourront voir les combats tout autour de la zone de combat où les combattants s'affronterontt sous les yeux de l'arbitre.

**Similation :** classe qui simule un combat entre 2 combattants.

**TestSimulation :** classe qui met en place un tournoi, c'est à dire un enchainement de combat où les vainqueurs s'affronteront jusqu'à ce qu'il n'en reste qu'un.

*Schéma UML des classes vision fournisseur (dessin "à la main" scanné ou photo acceptés)*



| |
|---|
| Schéma UML en plus lisible dans le dossier |

| *Checklist des contraintes prises en compte:* | *Nom(s) des classe(s) correspondante(s)* |
|---|---|
| Classe contenant un tableau ou une ArrayList | Classe Arene |
| Classe avec membres et méthodes statiques | Classe Fighter |
| Classe abstraite et méthode abstraite | Classe Personne |
| Interface | Classe Technique |
| Classe avec un constructeur par copie ou clone() | Classe Spectateur |
| Définition de classe étendant Exception | Classe PV_negatif |
| Gestion des exceptions | Classe Fighter |
| Utilisation du pattern singleton | Classe Arbitre |

*Présentation de votre projet (max. 2 pages) : texte libre expliquant en quoi consiste votre projet.*

Pour ce projet on avait pour objectif de créer notre propre sport. On a donc décidé de créer notre propre art martial qui est le regroupement de tous les arts martiaux.
Tous ceux qui s'intéressent un peu aux sports de combat auraient aimé voir des affrontements entre combattants avec des techniques de combat totalement différentes l'une de l'autre. Par exemple on assistera jamais à un combat entre un boxeur et un judoka car ils ne pourront s'affronter soit qu'avec les règles de la Boxe soit qu'avec les règles du Judo.
En effet notre projet réalise le rêve de tous ces passionnés de sport de combat et consiste à faire se battre 2 combattants avec des arts martiaux totalement différents avec toutes les règles de tous les arts martiaux regroupées. C'est en quelque sorte un combat sans règles.

Dans ce projet on a diférencié les combattants selon leur puissance qui varie entre 1 et 5 étoiles (1 étoile = 0 attaque spéciale, 2 étoiles = 1 attaques spéciales, 3 étoiles = 2 attaques spéciales, 4 étoiles = 3 attaques spéciales, 5 étoiles = 3 attaques spéciales + une technique) et en 2 catégories (c'est à dire 2 classes) : Legende et Moderne.

Moderne désigne un combattant champion de son art martial c'est à dire le plus fort dans sa catégorie. Ils ont pour maximum de puissance 4 étoiles. Ils démarrent tous avec 100pv.
Legende désigne un combattant legendaire ou fictif très fort. Ce sont les seules pouvant atteindre 5 étoiles de puissance donc les seules à posséder une technique. Eux démarrent avec 120pv chacun.

Pour ce projet nous avons mis en place 8 combattants au total : 2 Legende et 6 Moderne. Nous avons choisi seulement 2 Legende pour que ce soit équitable avec Moderne qui ont des stats plus faibles.
La liste des combattants Legende ainsi que leur art martial et leur nombre d'étoile →
-Hercule : Lutte greco-romaine/5 étoiles
-Madara Uchiwa : Ninjutsu/5 étoiles

La liste des combattants Moderne ainsi que leur art martial et leur nombre d'étoile →
-Bruce Lee : Kung Fu/4 étoiles
-Chuck Norris : Karaté/3 étoiles
-Mike Tyson : Boxe/3 étoiles
-Hakuho Sho : Sumo/3 étoiles
-Teddy Riner : Judo/2 étoiles
-Conor McGregor : MMA/2 étoiles

Tous les types de combats sont possibles, c'est à dire Legende vs Legende – Legende vs Moderne – Moderne vs Moderne.

Ils s'affronteront dans une arene (tableau de 5*4) créée via la classe Arene qui est composée en premier lieu de gradins qui entoure toute la zone de combat (sauf une case pour l'arbitre) et sont remplis par les spectateurs. Ensuite vient la zone de combat qui est le seule endroit où peuvent se déplacer les combattants.

L'objectif est de déterminer lors d'un tournoi quel est l'art martial le plus fort. Le gagnant de ce tournoi se verra donc être le plus fort de tous les combattants et son art martial le meilleur.

Notre projet répond aussi à la question suivante : Un combattant Moderne pourrai-t-il battre un combattant légendaire ou fictif ?
Prejugé ou véridique ? Pour y répondre il suffit de lancer la simulation de notre programme !

*Copier / coller vos classes et interfaces à partir d'ici :*

**Personne :**

```
public abstract class Personne{

	protected int posx;
	protected int posy;
	protected String nom;

	public Personne(int posx,int posy,String nom){
		this.posx = posx;
		this.posy = posy;
		this.nom = nom;
	}


	public void setPosition( int posx, int posy) { // Méthode permettant de donner une
nouvelle position à la personne
	this.posx = posx;
	this.posy = posy;
  }

   public void Positioninit() { // Position de la personne lorsque celle ci n'est pas dans l'arene (hors
de celle ci)
	this.posx = -1;
	this.posy = -1;
    }




	public abstract void action();//Méthode abstract permettant une action différente selon la
personne



	public String get_nom(){ //Méthode permettant d'obtenir le nom de la personne
	return this.nom;

   }

}
```

**Arbitre :**

```
public class Arbitre extends Personne{ // il n'y a qu'un arbitre, il est la pour marquer le début et la fin du tournoi

	private String nom;
	private int posx;
	private int posy;


	public Arbitre(int posx,int posy,String nom){
		super(posx,posy,nom);
	}


	public String toString(){
		return "Je suis un arbitre";
	}

	public void action(){
		System.out.println("L'arbitre siffle le début du combat");

	}
}
```

**Spectateur :**

```
public class Spectateur extends Personne{ //Les Spectateur sont autour du ring pour suporter les Fighter


   private String nom;
	private int posx;
	private int posy;

	public Spectateur(int posx,int posy, String nom){
		super(posx,posy,nom);

	}

	// constructeur par copie
	public Spectateur(Spectateur s){
		super(s.posx,s.posy,s.nom);


	}

	public void action(){
		System.out.println("Le Spectateur applaudit");
```

```
	}

	public String toString(){
		return "Spectateur"; // cela permet de savoir qui est le personnage
	}

}
```

**Fighter :**

```
public abstract class Fighter extends Personne {// classe dont héritent tous les combattants



	protected int posx;//position du combattant sur la ligne
	protected int posy;//position du combattant sur la colonne

	protected String nom;
	protected int etoile; // Puisance du combattant allant de 1 à 5 étoiles (1 etoile = 0 attaque spéciale, 2 etoiles = 1 attaques spéciales, 3 etoiles = 2 attaques spéciales, 4 etoiles = 3 attaques spéciales, 5 etoiles = 3 attaques spéciales plus une technique)
	protected int pv; // Point de vie du combattant


	public Fighter(String nom, int etoile, int pv,int posx, int posy){
		super(posx,posy,nom);
		this.etoile=etoile;
		this.pv=pv;


	}



	public String toString(){
		return "";
	}


	public void action(){
		System.out.println("Je suis sur le ring");
	}



	public String SeDeplacer(){ //Méthode qui permet le déplacement du fighter dans l'aréne seulement dans la zone de combat

	this.posx=(int)(Math.random()*3)+1; //les Fighter peuvent seulement se deplacer sur le ring et non dans toute l'arene
```

```
        this.posy=(int)(Math.random()*2)+1;
         return "Il se déplace sur la case ("+posx+","+posy+") ";

    }

    public int get_posx(){ //Méthode permettant d'obtenir la position du fighter dans l'aréne coté ligne
        return posx;
    }

    public int get_posy(){ //Méthode permettant d'obtenir la position du fighter dans l'aréne coté colonne
        return posy;
    }

    public int getPv(){ //Méthode permettant d'obtenir les pv du fighter
                return pv;
        }

        public boolean estKO() {//Méthode qui return true si un combattant est mis K.O c'est à dire que ses pv sont égales à 0
       return pv <= 0;
    }



        public void setPv(int newpv){// Méthode permettant une mise à jour des PV de chaque Fighter
                this.pv = newpv;

        }

    public void Attaque(int degat_attaque){ //Méthode qui reduit les pv du fighter après avoir pris un coup de la part de son adversaire
        pv=(this.pv)-degat_attaque;
        double i=Math.random(); // ici on lance un Math.random pour, selon les probabilités, avoir certains commentaire sur la puissance d'une attaque lorsque celle-ci est lancé
        if(i<0.3){
                System.out.println("OUUUIIIIE! -"+degat_attaque+" pv ça doit faire mal");
        }
                else {
                if(0.4<= i && i<=0.6){
                        System.out.println("Ca c'était un coup bien placé");
                        }
        }

    }

    public abstract String Attaque_spe(Fighter f);

    public static boolean Esquive(){ //Méthode permettant de savoir si le fighter à réussi à esquiver l'attaque de son adversaire
```

```
        double i=Math.random();
        if(i>0.5){
                return true;
        }
        return false;
    }

        public static void pv_negatif(Fighter f1) throws PV_negatif{
                if (f1.getPv()<0){
                        f1.setPv(0);
                        throw new PV_negatif("les Pv ne peuvent pas etre inférieur à 0 !");

                }
                else {
                        System.out.println();
                }
        }

}
```

**Legende :**

```
public class Legende extends Fighter implements Technique{ // Legende désigne un combattant mythique/légendaire connu de tous pour sa force et ses exploits

                // Seuls les légendes possèdent une technique
        protected String nom;
        protected int etoile; // Puisance du combattant allant de 1 à 5 étoiles (1 etoile = 0 attaque spéciale, 2 etoiles = 1 attaques spéciales, 3 etoiles = 2 attaques spéciales, 4 etoiles = 3 attaques spéciales, 5 etoiles = 3 attaques spéciales plus une technique)
        protected int pv; // Point de vie du combattant
        protected int posx;//position du combattant sur la ligne
        protected int posy;//position du combattant sur la colonne


        public Legende(String nom, int etoile, int pv,int posx, int posy){
                super(nom,etoile,pv,posx,posy);

        }



        public String toString(){
                return super.toString()+"";
        }

        public void technique(){
                System.out.println("Le combattant légendaire utilise sa technique");
        }
```

```
	@Override // La méthode étant abstract dans Fighter
	public String Attaque_spe(Fighter f){
		return " ";
	}

}
```

**Hercule :**

```
public class Hercule extends Legende{ //Héros de la mythologie greco-romaine représentant la force à l'état pur. Art martial : Lutte greco-romaine



	private String nom_art;//Nom de l'art martial pratiqué par le combattant
	private String nom;
	private int etoile; // Puisance du combattant allant de 1 à 5 étoiles (1 etoile = 0 attaque spéciale, 2 etoiles = 1 attaques spéciales, 3 etoiles = 2 attaques spéciales, 4 etoiles = 3 attaques spéciales, 5 etoiles = 3 attaques spéciales + 1 technique)
	private int posx;//position du combattant sur la ligne
	private int posy;//position du combattant sur la colonne
	private int pv;//Etant donné que Hercule posséde une Technique qui lui permet de se régenerer on lui fixe une limite de pv à ne pas dépasser pour ne pas avoir la possibilité d'avoir des pv à "l'infini"
	private int degat_att1;// Degat de l'attaque numero 1 du combattant
	private int degat_att2;
	private int degat_att3;



	public Hercule(String nom, int etoile, int pv,int posx, int posy){
		super(nom,etoile,pv,posx,posy);
		degat_att1=30;
		degat_att2=40;
		degat_att3=40;
	}



	public String toString(){
		return super.toString()+"Hercule";
	}

	public String getArt_martial(){ // Retourne le nom de l'art martial pratiqué par le combattant
		return this.nom_art;
	}



	public String Attaque_spe(Fighter f){ // f désigne son adversaire, cette méthode permet de simuler des attaques et d'enlever des pv à l'adversaire
	int i = (int)(Math.random()*2);
```

```
      switch (i){
         case 0:
         f.setPv(f.getPv()-degat_att1);
            return super.Attaque_spe(f)+"Hercule charge son poing avec sa Force du lion"; //coup de poing dévastateur
         case 1:
         f.setPv(f.getPv()-degat_att2);
            return super.Attaque_spe(f)+"Hercule fait appel à son père Zeus avec son Pouvoir des Dieux pour envoyer un éclair sur son adversaire"; //Zeus(son père) envoie un éclair sur l'adversaire
         case 2:
         f.setPv(f.getPv()-degat_att3);
            return super.Attaque_spe(f)+"Les coups de Hercule deviennent de plus en plus puissants grace aux 12 travaux"; //Durcicement du corps de Hercule grace à son dur entrainement

      }
                  return super.Attaque_spe(f)+"l'attaque a échoué";
   }

         public void technique(){ // Hercule possède une technique car il hérite de Legende
                  System.out.println("Hercule durcit son corps grace aux 12 travaux");//Les 12 travaux: -5 de dégat pour chaque attaque de l'adversaire (utilisable 1 fois). Les 12 travaux ont été tellement éprouvant que son corps s'est endurcis
         }

}
```

**Madara_Uchiwa :**

```
public class Madara_Uchiwa extends Legende{ //Combattant/ninja légendaire dans Naruto ( Art martial : Ninjutsu)



         private String nom_art;//Nom de l'art martial pratiqué par le combattant
         private String nom;
         private int etoile;// Puisance du combattant allant de 1 à 5 étoiles (1 etoile = 0 attaque spéciale, 2 etoiles = 1 attaques spéciales, 3 etoiles = 2 attaques spéciales, 4 etoiles = 3 attaques spéciales, 5 etoiles = 3 attaques spéciales plus une technique)
         private int posx;//position du combattant sur la ligne
         private int posy;//position du combattant sur la colonne
         private int degat_att1;// Degat de l'attaque numero 1 du combattant
         private int degat_att2;
         private int degat_att3;
         private int pv ;// Point de vie du combattant



         public Madara_Uchiwa(String nom, int etoile, int pv,int posx, int posy){
                  super(nom,etoile,pv,posx,posy);
                  degat_att1=40;
                  degat_att2=35;
                  degat_att3=30;
```

```
        }


        public String toString(){
                return super.toString()+"Madara Uchiwa";
        }

        public String getArt_martial(){ // Retourne le nom de l'art martial pratiqué par le
combattant
                return this.nom_art;
        }

        public String Attaque_spe(Fighter f){ // f désigne son adversaire, cette méthode permet de
simuler des attaques et d'enlever des pv à l'adversaire
        int i = (int)(Math.random()*2);
      switch (i){
        case 0:
        f.setPv(f.getPv()-degat_att1);
            return super.Attaque_spe(f)+"Madara plonge son adversaire dans un Genjutsu : Izanagi
et lui fais souffrir le martyre"; //Izanagi: Genjutsu(technique d'illusion) qui fait souffrir son
adversaire
        case 1:
        f.setPv(f.getPv()-degat_att2);
            return super.Attaque_spe(f)+"Madara lance un Katon Embrasement Suprême";
//Immense mur de flamme qui ne laisse aucune chance à son adversaire
        case 2:
        f.setPv(f.getPv()-degat_att3);
            return super.Attaque_spe(f)+"Madara développe son oeil et devient un Mangekyo
Sharingan et asséne de coup son adversaire"; //Mangekyo Sharingan: Dévelopement de l'oeil du
combattant qui lui permet d'assenir des coups précis qui touchent les points vitaux de l'adversaire

      }
                return super.Attaque_spe(f)+"l'attaque a échoué";
   }

        public void technique(){ // Madara possède une technique car il hérite de Legende
                System.out.println("Madara fais apparaitre Susano et tranche son adversaire");
//Susano: Fais apparaitre une gigantesque armure munit d'une énorme épée capable de tout
trancher sur son passage
        }

}
```

**Moderne :**

```
public class Moderne extends Fighter{ //Moderne désigne un combattant humain connu pour sa maitrise des arts martiaux
	protected String nom;
	protected int etoile; // Puisance du combattant allant de 1 à 5 étoiles (1 etoile = 0 attaque spéciale, 2 etoiles = 1 attaques spéciales, 3 etoiles = 2 attaques spéciales, 4 etoiles = 3 attaques spéciales, 5 etoiles = 3 attaques spéciales plus une technique)
	protected int pv; // Point de vie du combattant
	protected int posx;//position du combattant sur la ligne
	protected int posy;//position du combattant sur la colonne

	public Moderne(String nom, int etoile, int pv,int posx, int posy){
		super(nom,etoile,pv,posx,posy);

	}
	public String toString(){
		return super.toString()+"";
	}

	@Override
	public String Attaque_spe(Fighter f){ // Méthode étant abstract dans Fighter
		return " ";
	}

}
```

**Bruce_Lee :**

```
public class Bruce_Lee extends Moderne{ //Combattant représentant l'art martial : Kung-Fu

	private String nom_art;//Nom de l'art martial pratiqué par le combattant
	private String nom;
	private int etoile; // Puisance du combattant allant de 1 à 5 étoiles (1 etoile = 0 attaque spéciale, 2 etoiles = 1 attaques spéciales, 3 etoiles = 2 attaques spéciales, 4 etoiles = 3 attaques spéciales, 5 etoiles = 3 attaques spéciales plus une technique)
	private int posx;//position du combattant sur la ligne
	private int posy;//position du combattant sur la colonne
	private int degat_att1;// Degat de l'attaque numero 1 du combattant
	private int degat_att2;
	private int degat_att3;
	private int pv ;// Point de vie du combattant


	public Bruce_Lee(String nom, int etoile, int pv,int posx, int posy){
		super(nom,etoile,pv,posx,posy);
		this.degat_att1=30;
		this.degat_att2=30;
		this.degat_att3=35;
	}
```

```
	public String toString(){
		return super.toString()+"Bruce Lee";
	}


	public String getArt_martial(){ // Retourne le nom de l'art martial pratiqué par le combattant
		return this.nom_art;
	}


	public String Attaque_spe(Fighter f){ // f désigne son adversaire, cette méthode permet de simuler des attaques et d'enlever des pv à l'adversaire
	int i = (int)(Math.random()*2);
       switch (i){
	   case 0:
	   f.setPv(f.getPv()-degat_att1);
		return super.Attaque_spe(f)+"Bruce Lee attaque avec son Stamp Kick"; //le Stamp Kick: Un stomp est une frappe vers le bas avec le talon du pied depuis la position debout, et est généralement dirigée vers la tête ou le corps d'un adversaire abattu.
	   case 1:
	   f.setPv(f.getPv()-degat_att2);
		return super.Attaque_spe(f)+"Bruce Lee saute et utlise SideKick"; //le SideKick: Semblable au coup de pied avant, la puissance du coup de pied latéral est générée par les hanches et le tronc
	   case 2:
	   f.setPv(f.getPv()-degat_att3);
		return super.Attaque_spe(f)+"Bruce Lee se concentre avant d'assener un One-Inch Punch"; //le One-Inch Punch: Le one inch punch est une technique de coup de poing des arts martiaux chinois réalisée à très courte distance

       }
		return "l'attaque a échoué";
	}

}
```

**Chuck_Norris :**

```
public class Chuck_Norris extends Moderne{//Combattant représentant l'art martial : Karaté (avant d'être acteur Chuck Norris était connu pour sa maîtrise de différents arts martiaux dont le karaté)


	private String nom_art;//Nom de l'art martial pratiqué par le combattant
	private String nom;
	private int etoile; // Puisance du combattant allant de 1 à 5 étoiles (1 etoile = 0 attaque spéciale, 2 etoiles = 1 attaques spéciales, 3 etoiles = 2 attaques spéciales, 4 etoiles = 3 attaques spéciales, 5 etoiles = 3 attaques spéciales plus une technique)
	private int posx;//position du combattant sur la ligne
	private int posy;//position du combattant sur la colonne
	private int degat_att1;// Degat de l'attaque numero 1 du combattant
	private int degat_att2;
	private int pv ;// Point de vie du combattant



	public Chuck_Norris(String nom, int etoile, int pv,int posx, int posy){
		super(nom,etoile,pv,posx,posy);
		degat_att1=25;
		degat_att2=30;
	}


	public String toString(){
		return super.toString()+"Chuck Norris";
	}

	public String getArt_martial(){ // Retourne le nom de l'art martial pratiqué par le combattant
		return this.nom_art;
	}

	public String Attaque_spe(Fighter f){ // f désigne son adversaire, cette méthode permet de simuler des attaques et d'enlever des pv à l'adversaire
	int i = (int)(Math.random()*1);
       switch (i){
	case 0:
	f.setPv(f.getPv()-degat_att1);
	    return super.Attaque_spe(f)+"Chuck Norris saute et utilise son attaque Chun Kuk Do"; //Mouvements d'attaque et de défense tiré du taekwondo, du judo et même du Jeet Kun Do qui consiste à utiliser le pied en air tout en sautant.
	case 1:
	f.setPv(f.getPv()-degat_att2);
	    return super.Attaque_spe(f)+"Chuck Norris se concentre et met en place son Chuck Norris System"; //Style avec une base traditionnelle profondément enracinée qui lui permet une concentration extreme pour des coups plus puissants.
```

```
    }
            return super.Attaque_spe(f)+"l'attaque a échoué";
  }

}
```

**Mike_Tyson :**

```
public class Mike_Tyson extends Moderne{ //Combattant représentant l'art martial : Boxe

        private String nom_art;//Nom de l'art martial pratiqué par le combattant
        private String nom;
        private int etoile; // Puisance du combattant allant de 1 à 5 étoiles (1 etoile = 0 attaque spéciale, 2 etoiles = 1 attaques spéciales, 3 etoiles = 2 attaques spéciales, 4 etoiles = 3 attaques spéciales, 5 etoiles = 3 attaques spéciales plus une technique)
        private int posx;//position du combattant sur la ligne
        private int posy;//position du combattant sur la colonne
        private int degat_att1;// Degat de l'attaque numero 1 du combattant
        private int degat_att2;
        private int pv ;// Point de vie du combattant


        public Mike_Tyson(String nom, int etoile, int pv,int posx, int posy){
                super(nom,etoile,pv,posx,posy);
                degat_att1=20;
                degat_att2=40;
        }



        public String toString(){
                return super.toString()+"Mike Tyson";
        }


        public String getArt_martial(){ // Retourne le nom de l'art martial pratiqué par le combattant
                return this.nom_art;
        }

        public String Attaque_spe(Fighter f){ // f désigne son adversaire, cette méthode permet de simuler des attaques et d'enlever des pv à l'adversaire
        int i = (int)(Math.random()*1);
       switch (i){
        case 0:
        f.setPv(f.getPv()-degat_att1);
            return super.Attaque_spe(f)+"Mike Tyson se rapproche de son adversaire et lui arrache l'oreille avec sa célébre Morsure de l'oreille"; //Arrache un morceau d'oreille de son adversaire (Triche qu'il a réellement commis lors d'un combat)
        case 1:
```

```
        f.setPv(f.getPv()-degat_att2);
            return super.Attaque_spe(f)+"Mike Tyson arme son poing et déclenche un Iron Mike"; //frappe explosive de Tyson était en grande partie due au fait de s'accroupir juste avant de lancer un crochet ou un uppercut: cela permettait au «ressort» de ses jambes d'ajouter de la puissance au coup de poing.

      }
                return super.Attaque_spe(f)+"l'attaque a échoué";
   }

}
```

**Hakuho_Sho :**

```
public class Hakuho_Sho extends Moderne{//Combattant représentant l'art martial : Sumo



        private String nom_art;//Nom de l'art martial pratiqué par le combattant
        private String nom;
        private int etoile; // Puisance du combattant allant de 1 à 5 étoiles (1 etoile = 0 attaque spéciale, 2 etoiles = 1 attaques spéciales, 3 etoiles = 2 attaques spéciales, 4 etoiles = 3 attaques spéciales, 5 etoiles = 3 attaques spéciales plus une technique)
        private int posx;//position du combattant sur la ligne
        private int posy;//position du combattant sur la colonne
        private int degat_att1;// Degat de l'attaque numero 1 du combattant
        private int degat_att2;
        private int pv ;// Point de vie du combattant


        public Hakuho_Sho(String nom, int etoile, int pv,int posx, int posy){
                super(nom,etoile,pv,posx,posy);
                degat_att1=25;
                degat_att2=35;
        }


        public String toString(){
                return super.toString()+"Hakuho_Sho";
        }


        public String getArt_martial(){ // Retourne le nom de l'art martial pratiqué par le combattant
                return this.nom_art;
        }

        public String Attaque_spe(Fighter f){ // f désigne son adversaire, cette méthode permet de simuler des attaques et d'enlever des pv à l'adversaire
        int i = (int)(Math.random()*1);
      switch (i){
        case 0:
```

```
        f.setPv(f.getPv()-degat_att1);
            return super.Attaque_spe(f)+"Hakuho Sho cours vers son adversaire et éxécute le Ashitori"; //Attrape le bas de la jambre de son opposant et relève celle-ci avec les 2 mains pour mettre son adversaire au sol
        case 1:
        f.setPv(f.getPv()-degat_att2);
            return super.Attaque_spe(f)+"Hakuho Sho se met en position et déclenche un Izori"; //Le combattant ploonge sous la charge de son opposant puis lui attrape les genoux pour le jeter au sol
      }
                return super.Attaque_spe(f)+"l'attaque a échoué";
   }


}
```

**Teddy_Riner :**

```
public class Teddy_Riner extends Moderne{ //Combattant représentant l'art martial : Judo



        private String nom_art;//Nom de l'art martial pratiqué par le combattant
        private String nom;
        private int etoile; // Puisance du combattant allant de 1 à 5 étoiles (1 etoile = 0 attaque spéciale, 2 etoiles = 1 attaques spéciales, 3 etoiles = 2 attaques spéciales, 4 etoiles = 3 attaques spéciales, 5 etoiles = 3 attaques spéciales plus une technique)
        private int posx;//position du combattant sur la ligne
        private int posy;//position du combattant sur la colonne
        private int degat_att;// Degat de l'attaque du combattant
        private int pv;// Point de vie du combattant

        public Teddy_Riner(String nom, int etoile, int pv,int posx, int posy){
                super(nom,etoile,pv,posx,posy);
                this.degat_att=30;

        }



        public String toString(){
                return super.toString()+"Teddy Riner";
        }

        public String getArt_martial(){ // Retourne le nom de l'art martial pratiqué par le combattant
                return this.nom_art;
        }

        public String Attaque_spe(Fighter f){ // f désigne son adversaire, cette méthode permet de simuler des attaques et d'enlever des pv à l'adversaire
                f.setPv(f.getPv()-degat_att);
```

```
                return super.Attaque_spe(f)+"Teddy Riner enclenche un Ippon à son adversaire à l'aide son corps massif";//Ippon:Retourne son adversaire et le fracasse contre le sol

        }
}
```

**Conor_McGregor :**

```
public class Conor_McGregor extends Moderne{//Combattant représentant l'art martial : MMA


        private String nom_art;//Nom de l'art martial pratiqué par le combattant
        private String nom;
        private int etoile; // Puisance du combattant allant de 1 à 5 étoiles (1 etoile = 0 attaque spéciale, 2 etoiles = 1 attaques spéciales, 3 etoiles = 2 attaques spéciales, 4 etoiles = 3 attaques spéciales, 5 etoiles = 3 attaques spéciales plus une technique)
        private int posx;//position du combattant sur la ligne
        private int posy;//position du combattant sur la colonne
        private int degat_att;// Degat de l'attaque du combattant
        private int pv ;// Point de vie du combattant


        public Conor_McGregor(String nom, int etoile, int pv,int posx, int posy){
                super(nom,etoile,pv,posx,posy);
                degat_att=25;

        }


        public String toString(){
                return super.toString()+"Conor McGregor";
        }

        public String getArt_martial(){ // Retourne le nom de l'art martial pratiqué par le combattant
                return this.nom_art;
        }

        public String Attaque_spe(Fighter f){ // f désigne son adversaire, cette méthode permet de simuler des attaques et d'enlever des pv à l'adversaire
                f.setPv(f.getPv()-degat_att);
                return super.Attaque_spe(f)+"McGregor s'avance et attaque avec son Spinning Back Kick";//Technique de pied arrière en rotation qui consiste à avancer avec la jambe de tête, pivoter et tourner pour lancer la jambe arrière dans le corps de l'adversaire.
        }

}
```

**Technique :**

```
public interface Technique{ //Interface permettant à seulement certains Fighter de posséder une Technique

	public void technique();

}
```

**PV_negatif :**

```
public class PV_negatif extends Exception{ // déclenche une exception sur les pv négatifs

   public PV_negatif(String message){
	super(message);
   }

}
```

**Arene :**

```
public class Arene{//Terrain où se dérouleront les combats

	private final Personne[][] arene;


	private int nb_colonne;
	private int nb_ligne;

	public Arene(int nb_ligne,int nb_colonne){
		this.nb_ligne = nb_ligne;
		this.nb_colonne = nb_colonne;
		this.arene = new Personne[this.nb_ligne][this.nb_colonne];


	}

	public boolean sontValides(int i, int j) { // Méthode permettant de savoir si la case choisie est bien dans l'arene
	  return i >= 0 && i < this.nb_ligne && j >= 0 && j < this.nb_colonne;
   }


	public Personne getcase(int ligne, int colonne){ //Méthode permettant d'obtenir le contenu d'une case choisi
		return arene[ligne][colonne];

	}
```

```
        public boolean case_est_vide(int ligne, int colonne){ //Méthode permettant de savoir si la case demandée est vide

                return arene[ligne][colonne]==null;

        }

        public boolean setCase(int i, int j, Personne p) { // Méthode permettant de placer une personne sur l'arene
      if (this.sontValides(i, j)) {
         if (this.arene[i][j] != null) {
            this.arene[i][j].Positioninit();
         }

         this.arene[i][j] = p;
         p.setPosition(i, j);
         return true;
      } else {
         return false;
      }
   }

   public Personne actualiser(int i, int j) { //Méthode qui permet d'actualiser la position des combattant dans l'arene
      if (this.sontValides(i,j) && this.arene[i][j] != null) {
         Personne p = this.arene[i][j];
         p.Positioninit();
         this.arene[i][j] = null;
         return p;
      } else {
         return null;
      }
   }




        public void affichage() { // construction du visuel du tableau pour l'afficher
      String s1 = "";
      String s2 = ":"; // caractères permettant de délimiter les cases
      String s3 = "";
                String s4= "";
      String s5 = "";
      String s6 = "";
      String s7 = ":";
      int i;

      for(i= 0; i < 20; i++) {
         s3 = s3 + "-";
```

```
        }

        for(i = 0; i < this.nb_colonne; i++) {
            s2 = s2 + s3 + ":";
        }

                    for(i= 0; i < 20; i++) {
            s5 = s5 + " ";
        }

        for(i= 0; i < nb_colonne-2; i++) {
            s6 = s6 + s5;
        }

        for(i= 0; i < nb_colonne-3; i++) {
            s6 = s6 + " ";
        }

        s4 = s7 + s3 + s7 + s6 + s7 + s3 + s7;
        s4 = s4 + "\n";

        for(i= 0; i < 20; i++) {
            s3 = s3 + " ";
        }
        s2 = s2 + "\n";
        s1 = s2;

                    for(int j = 0; j < this.nb_colonne; j++) {
                if (this.arene[0][j] == null) {
                    s1 = s1 + " " + String.format("%-20s", " "); // format des cases
                } else {
                    s1 = s1 + " " + this.nb_lettres(this.arene[0][j].nom); // affichage du nom de la
personne présente sur la case
                }
            }

                    s1 = s1 + " \n" + s2;

        for(i = 1; i < this.nb_ligne-2; i++) {
            for(int j = 0; j < this.nb_colonne; j++) {
                if (this.arene[i][j] == null) {
                    s1 = s1 + " " + String.format("%-20s", " "); // format des cases
                } else {
                    s1 = s1 + " " + this.nb_lettres(this.arene[i][j].nom); // affichage du nom de la
personne présente sur la case
                }
            }

            s1 = s1 + " \n" + s4;
        }
                    for(int k=0; k<2 ;k++) {
```

```
                        for(int j = 0; j < this.nb_colonne; j++) {
            if (this.arene[this.nb_ligne-2+k][j] == null) {
                s1 = s1 + " " + String.format("%-20s", " "); // format des cases
            } else {
                s1 = s1 + " " + this.nb_lettres(this.arene[this.nb_ligne-2+k][j].nom); // affichage du
nom de la personne présente sur la case
            }
        }

                s1 = s1 + " \n" + s2;
                 }

      System.out.println(s1);
   }


 private String nb_lettres(String s1) { // méthode permettant l'affichage d'un nombre de lettres
précis du nom de la personne
      String s2 = String.format("%-20s", s1);
      return s2.substring(0, 20);
   }

}
```

**Similation :**

```
public class Simulation{
        private static Fighter f1; //les deux combattants
        private static Fighter f2;
        private static Arene arene; // arene sur laquelle on travaille
        private static Arbitre arbitre; //l'arbitre


        public Simulation(Fighter f1, Fighter f2, Arbitre arbitre){
                this.f1 = f1;
                this.f2 = f2;
                this.arbitre = arbitre;
                this.arene = new Arene(5,4);
                Spectateur s1 = new Spectateur(0,0,"spectateur");
                Spectateur s2 = new Spectateur(s1);

                this.arene.setCase(0,0, s1); //placement des spectateurs et de l'arbitre sur l'arene
(les spectateur sont tous les memes)
                this.arene.setCase(0,1, s1);
                this.arene.setCase(0,2, s1);
                this.arene.setCase(0,3, s1);
                this.arene.setCase(1,0, s1);
                this.arene.setCase(0,0, s1);
                this.arene.setCase(3,0, s1);
```

```
		this.arene.setCase(4,0, s1);
		this.arene.setCase(4,1, s1);
		this.arene.setCase(4,2, s1);
		this.arene.setCase(4,3, s1);
		this.arene.setCase(1,3, s1);
		this.arene.setCase(2,3, s1);
		this.arene.setCase(3,3, s1);
		this.arene.setCase(2,0,new Arbitre(3,1,"arbitre"));

	}



	public static Fighter Combat(Fighter f1, Fighter f2){ // simulation d'un combat

		arbitre.action();
		for(int i=1;i<4;i++){
			for (int j =1;j<3;j++){
				arene.actualiser(i,j); // on vide les cases du ring
			}
		}


		arene.setCase(1,1, f1); // positions de depart des combattants
		arene.setCase(3,2, f2);
		arene.affichage();

		arene.actualiser(1,1);
		arene.actualiser(3,2);



		while(f1.getPv()>0 && f2.getPv()>0){ // tant que les deux Fighter ont des Pv, ils
continuent à combattre


			arene.actualiser(f1.posx,f1.posy);//on vide la case ou le Fighter était avant
sont déplacement
			f1.SeDeplacer();// les Fighter se déplace chacun leur tour
			arene.setCase(f1.posx,f1.posy,f1);

			System.out.println();
			System.out.println();

			arene.affichage();// affichage de l'arene avec la nouvelle position des
Fighter

			System.out.println();
			System.out.println();

			if (f1.posx==f2.posx && f1.posy==f2.posy){ // l'attaque a lieu seulement si
```

les Fighter sont sur la meme case , dans ce cas la c'est seulement le nom du Fighter qui lance sont attaque qui apparait sur l'affichage de l'arene

```
                        if (f2.Esquive()==true){ //l attaque peut etre esquivé par le Fighter adverse
                                System.out.println(f1.get_nom()+" execute son coup mais "+f2.get_nom()+ " l'esquive de justesse !");
                        }
                        else{
                                if (f1 instanceof Legende && f1.getPv()<=40){ // si le Fighter est une légende et que ses pv sont bas alors il peut utiliser sa technique, seulement celle ci n'a qu'une chance sur deux d'etre lancé
                                        double i=Math.random();
                                if(i<0.5){
                                        f2.setPv(f2.getPv()-45); // une technique enleve 45 pv à son adversaire

                                        ((Legende) f1).technique();
                                        System.out.println("les pv de "+f2.get_nom()+" descendent à "+f2.getPv());
                                }
                                else{
                                        System.out.println("Il n'a pas pu utiliser sa technique légendaire");
                                }

                                }
                                else{ // si l'adversaire n'a pas esquivé alors il subit une attaque
                                System.out.println(f1.Attaque_spe(f2));

                                try{
                                        Fighter.pv_negatif(f2);
                                                }catch(PV_negatif e){
                                                System.out.println(e.getMessage());
                                        }



                                System.out.println("les pv de "+f2.get_nom()+" descendent à "+f2.getPv());
                        }
                }



                }
                else{ // si les deux Fighter n'étaient pas sur la meme case c'est alors au deuxième de se déplacer, son tour se passe comme celui du premier
                        arene.actualiser(f2.posx,f2.posy);
                        f2.SeDeplacer();
```

```
                        arene.setCase(f2.posx,f2.posy,f2);

                        System.out.println();
                        System.out.println();
                        arene.affichage();
                        System.out.println();
                        System.out.println();

                        if (f2.posx==f1.posx && f1.posy==f2.posy){
                        if (f1.Esquive()==true){
                                System.out.println(f2.get_nom()+" execute son coup mais
"+f1.get_nom()+ " l'esquive de justesse !");
                        }
                        else{
                                if (f2 instanceof Legende && f2.getPv()<=40){
                                        double i=Math.random();
                                if(i<0.5){
                                        f1.setPv(f1.getPv()-45);
                                        ((Legende) f2).technique();
                                        System.out.println("les pv de "+f1.get_nom()+"
descendent à "+f1.getPv());
                                }
                                else{
                                        System.out.println("Il n'a pas pu utiliser sa technique
légendaire");
                                }

                                }
                                else{
                                System.out.println(f2.Attaque_spe(f1));
                                try{
                                        Fighter.pv_negatif(f1);
                                                }catch(PV_negatif e){
                                                System.out.println(e.getMessage());
                                        }
                                System.out.println("les pv de "+f1.get_nom()+" descendent
à "+f1.getPv());
                                }
                        }



                        }


                        else{
                                System.out.println("les combattants ne sont pas face à face
et ne peuvent donc pas se mettre de coups");
                        }
                }
```

```
                }

                //lorsque les Pv de l'un des Fighter sont à zero alors le vainqueur est affiché
                if(f1.estKO()){
                        System.out.println("L'arbitre siffle l'arret du combat! "+f2.get_nom()+"
l'emporte par K.O");
                  f2.setPv(100);//les pv de chaque fighter vainqueur se remettent à 100, meme ceux
des légendaires
                        return f2;// retourne le vainqueur du combat
                }
                else{
                        System.out.println("L'arbitre siffle l'arret du combat! "+f1.get_nom()+"
l'emporte par K.O");
                        f1.setPv(100);
                        return f1; // retourne le vainqueur du combat
                }


        }

}
```

**TestSimulation :**

```
import java.util.ArrayList;
import java.util.*;

public class TestSimulation{
        public static void main(String[] args) {

                Bruce_Lee Bruce_Lee = new Bruce_Lee("Bruce Lee",4,100,4,3);
                Teddy_Riner Teddy_Riner = new Teddy_Riner("Teddy Riner",2,100,3,2);
                Hercule Hercule = new Hercule("Hercule",5,120,4,3);
                Chuck_Norris Chuck_Norris  = new Chuck_Norris("Chuck Norris",3,100,3,2);
                Conor_McGregor Conor_McGregor = new Conor_McGregor("Conor
McGregor",2,100,4,3);
                Madara_Uchiwa Madara_Uchiwa = new Madara_Uchiwa("Madara
Uchiwa",5,120,3,2);
                Hakuho_Sho Hakuho_Sho = new Hakuho_Sho("Hakuho Sho",3,100,4,3);
                Mike_Tyson Mike_Tyson = new Mike_Tyson("Mike Tyson",3,100,3,2);

                ArrayList<Fighter> arraylist = new ArrayList<Fighter>(); // création d'une
Arrayliste comportant tous les Fighter
   arraylist.add(Bruce_Lee);
   arraylist.add(Teddy_Riner);
   arraylist.add(Hercule);
   arraylist.add(Chuck_Norris);
                arraylist.add(Conor_McGregor);
   arraylist.add(Madara_Uchiwa);
   arraylist.add(Hakuho_Sho);
```

```
   arraylist.add(Mike_Tyson);

          for (int i =0; i<8; i++){
                try{
                      Fighter.pv_negatif(arraylist.get(i));
                }catch(PV_negatif e){
                      System.out.println(e.getMessage());
                }
          }


   Collections.shuffle(arraylist); //On mélange la liste de facon aléatoire pour que les combats soient déterminés aléatoires



                      System.out.println("-------------------------------------------------------------");
                      System.out.println("-                                                           -");
                      System.out.println("-              TIRAGE AU SORT DES COMBAT                    -");
                      System.out.println("-                                                           -");
                      System.out.println("-------------------------------------------------------------");

       for (int i =0; i<4; i++){
                                    int j = i+1;
          System.out.println("Ring "+j);
          System.out.println(arraylist.get(i*2));//Les Figher cote a cote dans la liste vont s'affronter (1VS2 , 3VS4 ...)
                    System.out.println("VS");
                    System.out.println(arraylist.get(i*2+1));
          System.out.println("");

       }

                      System.out.println("-------------------------------------------------------------");
                      System.out.println("-                                                           -");
                      System.out.println("-          DEBUT DES COMBAT : PREMIER TOUR                  -");
                      System.out.println("-                                                           -");
                      System.out.println("-------------------------------------------------------------");


                Arbitre A = new Arbitre(2,0,"arbitre");

                Simulation s0 = new Simulation(arraylist.get(0),arraylist.get(1),A);
                Simulation s1 = new Simulation(arraylist.get(2),arraylist.get(3),A);
                Simulation s2 = new Simulation(arraylist.get(4),arraylist.get(5),A);
                Simulation s3 = new Simulation(arraylist.get(6),arraylist.get(7),A);

                Simulation[] tab ={s0,s1,s2,s3};
                ArrayList<Fighter> arraylist2 = new ArrayList<Fighter>(); //on créer une nouvelle Arrayliste pour les vainqueurs de chaque combat
```

```
		for (int i =0; i<4; i++){
		int j = i+1;

		System.out.println("");
		System.out.println("");
	System.out.println("------------------------------------------------------------");
		System.out.println("-                                                          -");
		System.out.println("-                RING "+j+"                                -");
		System.out.println("-                                                          -");
		System.out.println("------------------------------------------------------------");
	System.out.println("");
	System.out.println("");
		arraylist2.add(tab[i].Combat(arraylist.get(2*i),arraylist.get(2*i+1))); //les
vainqueurs sont ajouté à la liste
		System.out.println("");
		System.out.println("");

		}



		Collections.shuffle(arraylist2); //on mélange la liste de vainqueur pour tirer au sort

			System.out.println("------------------------------------------------------------");
			System.out.println("-                                                          -");
			System.out.println("-              DEMI FINALE                                 -");
			System.out.println("-                                                          -");
			System.out.println("------------------------------------------------------------");

			Simulation s4 = new Simulation(arraylist2.get(0),arraylist2.get(1),A);
			Simulation s5 = new Simulation(arraylist2.get(2),arraylist2.get(3),A);
			Simulation[] tab2 ={s4,s5};
			ArrayList<Fighter> arraylist3 = new ArrayList<Fighter>(); //on créer une
nouvelle Arrayliste pour les vainqueurs de chaque combat

		for (int i =0; i<2; i++){
			int j = i+1;

			System.out.println("");
			System.out.println("");
		System.out.println("------------------------------------------------------------");
			System.out.println("-                                                          -");
			System.out.println("-                RING "+j+"                                -");
			System.out.println("-                                                          -");
			System.out.println("------------------------------------------------------------");
		System.out.println("");
		System.out.println("");
			arraylist3.add(tab2[i].Combat(arraylist2.get(2*i),arraylist2.get(2*i+1)));//
les vainqueurs sont ajouté à la liste
			System.out.println("");
```

```
                        System.out.println("");

                }

                        System.out.println("-------------------------------------------------------------");
                        System.out.println("-                                                           -");
                        System.out.println("-                    FINALE                                 -");
                        System.out.println("-                                                           -");
                        System.out.println("-------------------------------------------------------------");

                        System.out.println("Ring final");
    System.out.println(arraylist3.get(0));
                    System.out.println("VS");
                    System.out.println(arraylist3.get(1));
    System.out.println("");

    Simulation s6 = new Simulation(arraylist3.get(0),arraylist2.get(1),A);
    ArrayList<Fighter> arraylist4 = new ArrayList<Fighter>(); // on créer une nouvelle Arrayliste
pour le vainqueur du dernier combat

    System.out.println("");
                        System.out.println("");
                System.out.println("-------------------------------------------------------------");
                        System.out.println("-                                                           -");
                        System.out.println("-                    RING FINAL                             -");
                        System.out.println("-                                                           -");
                        System.out.println("-------------------------------------------------------------");
                System.out.println("");
                System.out.println("");
                        arraylist4.add(s6.Combat(arraylist3.get(0),arraylist3.get(1))); // La liste
comporte le vainqueur final
                        System.out.println("");
                        System.out.println("");
                        System.out.println("");
                        System.out.println("");
                        System.out.println("");

        System.out.println("-----------------------------------------------------------------------------");
                        System.out.println("-                                                           -");
                        System.out.println("-          LE GRAND VAINQUEUR EST
"+arraylist4.get(0)+"                    -");//le seul élément de la liste étant le vainqueur, il est
                        System.out.println("-                                                           -");
//affiché ici

        System.out.println("-----------------------------------------------------------------------------");


        }

}
```