Conception orientée objet

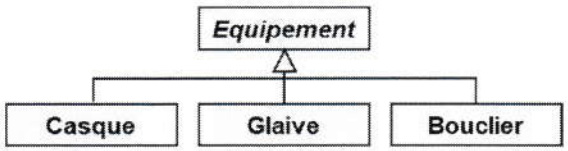
Application du cours 3 : Musée et rangement en tas

Cet énoncé vient en appui des diapositives du Cours.

I. Enoncé du TD – première partie : généricité

1. Classe RenseignementTrophee

```
public class RenseignementTrophee {
  private Gaulois proprietaire;
  private Equipement trophee;

  public RenseignementTrophee(Gaulois proprietaire, Equipement trophee) {
    this.proprietaire = proprietaire;
    this.trophee = trophee;
  }

  public Gaulois getProprietaire() {
    return proprietaire;
  }

  public Equipement getTrophee() {
    return trophee;
  }
}
```



2. Classe Paire

```
public class Paire<T, U> {
  private final T premier;
  private final U second;

  public Paire(T premier, U second) {
    this.premier = premier;
    this.second = second;
  }

  public T getPremier() {
    return premier;
  }

  public U getSecond() {
    return second;
  }
}
```

Nous souhaitons utiliser la classe générique « Paire<T, U> » afin de remplacer à terme la classe « Renseignement ».

Dans la classe « Test » nous avions des exemples de création d'objets de la classe « RenseignementTrophee ».

```
RenseignementTrophee asterixTrophee1 =
        new RenseignementTrophee(asterix, new Bouclier());
RenseignementTrophee asterixTrophee2 =
        new RenseignementTrophee(asterix, new Casque());
RenseignementTrophee obelixTrophee1 =
        new RenseignementTrophee(obelix, new Glaive());
```

Remplacer le type de ces objets par le type générique « Paire<T,U> »

Paire<Gaulois, Equipement> asterixTrophee1 =
new Paire<>(asterix, new Bouclier());

Paire<Gaulois, Equipement> asterixTrophee2 =
new Paire<>(asterix, new Casque());

Paire<Gaulois, Equipement> obelixTrophee1 =
new Paire<>(obelix, new Glaive());

Faire afficher le trophée (le glaive) qu'a ramené Obélix.

System.out.println(~~obelixTh~~ obelixTrophee1.getSecond());

II. Enoncé du TD – deuxième partie : itérateurs et classes itérables

1. Les Iterateurs

Obélix organise en tas les équipements récupérés sur les romains : un tas de casques, un tas de glaives, un tas de boucliers … Mais attention, il est très ordonné et ne les mélange pas !

1. Créer la classe « Tas » qui contient un tableau tasEquipement. Ce tableau est transmis par le constructeur. Un attribut nombreEquipement est initialisé à 0 et permet de connaitre le nombre d'équipements que contient le tas.
2. Ecrire la méthode *ajouterEquipement* qui ajoute un équipement au tableau tas.
3. Implémenter l'interface « *Iterator* ».

public class Tas <E extends Equipement>

{

//Attributs

```
private E[] tasEquipement;
private int nombreEquipement = 0;
```

//Constructeur

```
public Tas(E[] tas){
    tasEquipement = tas;
}
```

//Méthodes

public void ajouterEquipement(E equipement) {

```
tasEquipement[nombreEquipement] = equipement;
nombreEquipement++;
}
```

//Méthodes de l'interface Iterator

```
public E next() throws NoSuchElementException {
    if(hasNext()){
        E equipement = tasEquipement[indiceIterateur];
        indiceIterateur++;
        return equipement;
    }
    throw new NoSuchElementException();
}
```

```
public void remove() throws IllegalStateException {
    if(nombreEquipement < 1) {throws new IllegalStateException();}
    for(int i = indicateur - 1; i < nombreEquipement - 1; i++){
        tasEquipement[i] = tasEquipement[i+1];
    }
indiceIterateur--;
nombreEquipement--;
}

public boolean hasNext() {                    indiceIteratif
    return (nombreEquipement != 0 && nombreEquipement > 0);
}
```

Compléter le main de la classe « Test » afin qu'il affiche l'ensemble du tas.

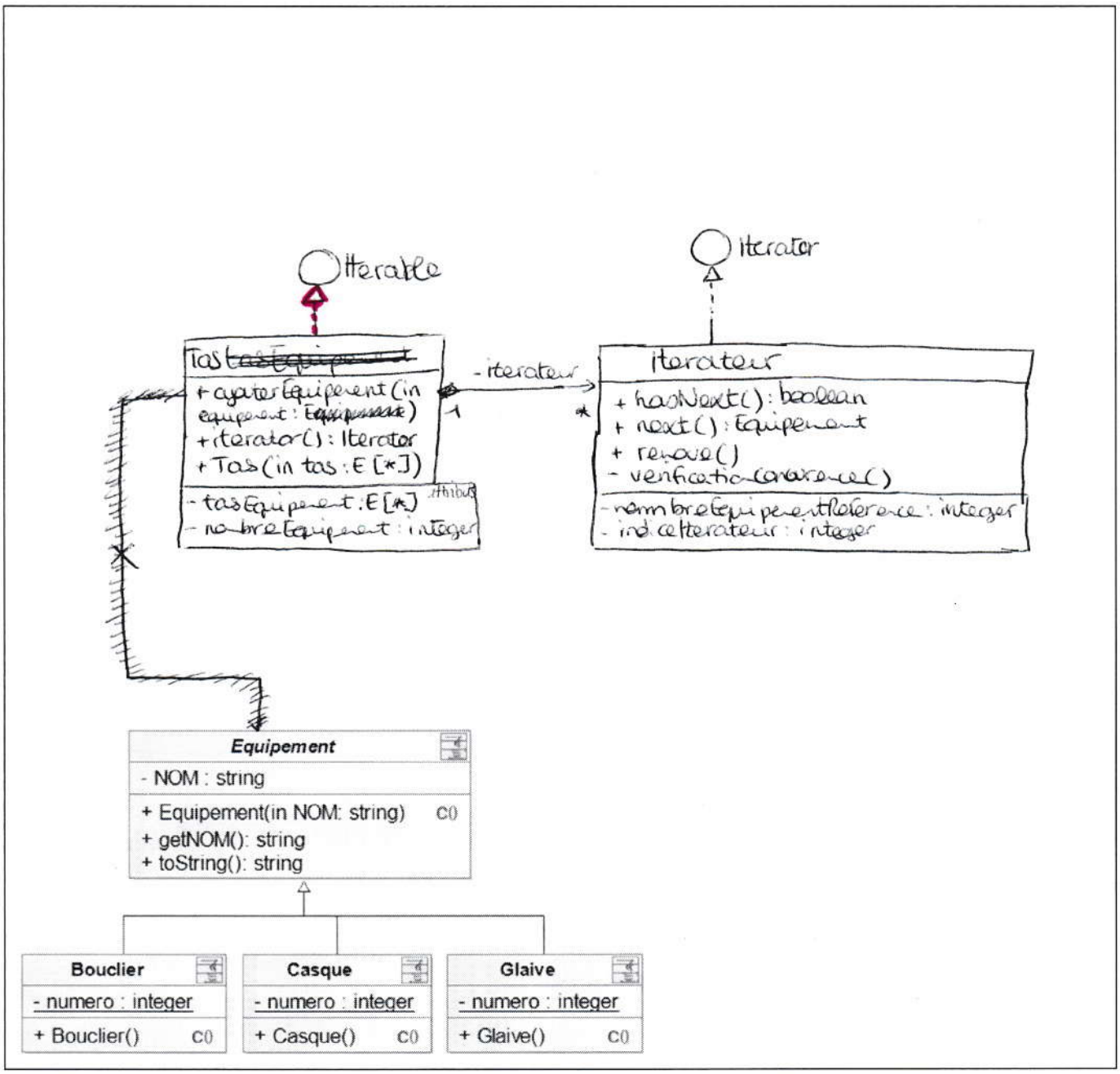
```
Glaive glaive1 = new Glaive();
Glaive glaive2 = new Glaive();
Glaive glaive3 = new Glaive();

Tas<Glaive> tasGlaive = new Tas<>(new Glaive[5]);
tasGlaive.ajouterEquipement(glaive1);
tasGlaive.ajouterEquipement(glaive2);
tasGlaive.ajouterEquipement(glaive3);

For(; tasGlaive.hasNext();){
    System.out.println(tasGlaive.next());
}
```

2. *Classe Iterable*

Notre classe « Tas » implémente l'interface « Iterator », mais nous pouvons le parcourir qu'une seule fois. Pour pouvoir le parcourir autant de fois que nécessaire, nous souhaitons l'améliorer afin qu'elle :

- soit itérable,
- soit générique pour avoir soit un tas de casques, soit un tas de glaives, soit un tas de boucliers ...
- possède la méthode *afficherTas* qui parcourt et affiche l'ensemble d'un tas.

Spécifier, à l'aide d'UML, la nouvelle conception.



Ecrire le code Java correspondant.

public class Tas <E extends Equipement> implements Iterable<E>
{

//Attributs

private E[] tasEquipement;
private int nombreEquipement=0;

//Constructeur

pubic Tas(E[] tas){
tasEquipement = tas;
}

//Méthodes

public void ajouterEquipement(E equipement) {
tasEquipement[nombreEquipement] = equipement;
nombreEquipement ++;
}

//Méthode de l'interface Iterable

pubic Iterator<E> iterator() {
return new Iterateur<>();
}

//Classe interne

private class Iterateur implements Iterator<E>{

//Attributs

private int nombreEquipementReference = nombreEquipement;
private int indiceIterateur=0;

//Méthode de vérification de la concurrence entre itérateurs

```
private void verificationConcurrence() throws ConcurrentModification
                                                -cationException {
    if(nombreEquipement != nombreEquipementReference){
        throw new ConcurrentModificationException
    }

}
```

//hasNext

```
public boolean hasNext throws ConcurrentModificationException,
                                                        {
    verificationConcurrence();
    return nombreEquipement != 0 && indiceIterateur < nombreEqui
                                                        pement

}
```

//next

```
public E next() throws ConcurrentModificationException,
NoSuchElementException
    verificationConcurrence();                          {
    if(hasNext()){
        E equipement = tabEquipement[indiceIterateur];
        indiceIterateur++;
        return

}
```

```
        //remove
```

(voir verso page 4) public void remove() throws ConcurrentModificationException {

```
        }
}}
```

Compléter le main de la classe « Test » afin qu'il :
- affiche l'ensemble du tas,
- supprime le premier glaive,

```
Glaive glaive1 = new Glaive(); Glaive glaive2 = new Glaive();
Glaive glaive3 = new Glaive();
Tas<Glaive> tasGlaive = new Tas<>(new Glaive[5]);
tasGlaive.ajouterEquipement(glaive1);
tasGlaive.ajouterEquipement(glaive2);
tasGlaive.ajouterEquipement(glaive3);
System.out.println("Affichage des trois glaives");
for(Glaive glaive: tasGlaive){
    System.out.println(glaive)
}
System.out.println("\nSupression du premier glaive");
Iterator<Glaive> iterGlaive = tasGlaive.iterator();
iterGlaive.next();
iterGlaive.remove();
for(Glaive glaive: tasGlaive){
    System.out.println(glaive);
}
```