LE DEVELOPPEMENT EN COUCHES AVEC JAVA SE

Module 1 – JAVA – Notions complémentaires

Objectifs

- Comprendre et implémenter les interfaces
- Comprendre et implémenter les collections
- Comprendre et implémenter les génériques

Définir une interface

```
[public] interface NomInterface
                        [extends NomSuperInterface1,
                        NomSuperInterface2, …] {
    //constantes
    [public][static] [final] type NOM_CONSTANTE = valeur;
    //méthodes abstraites
    [public] [abstract] type  methode(  liste_parametres );
}
```

Implémenter une interface

```
public interface Predateur {
   void chasse(Proie p);
}
```

```
public class Chat implements Predateur {
      @Override
      public void chasse(Proie p)  {
         …
      }

}
```

L'intérêt de l'interface

- Complémentaire à l'héritage :
  - Remplace l'héritage multiple
  - Permet de faire partager des fonctionnalités communes à des classes différentes
  - Vue comme un contrat imposé aux classes qui l'implémente
  - Vue comme un type

Les interfaces

Démonstration

Les ensembles

- Besoin de variables faisant référence à plusieurs éléments
- 3 solutions :
  - Utiliser un tableau

```
Personne[] equipe = new Personne[10];
```

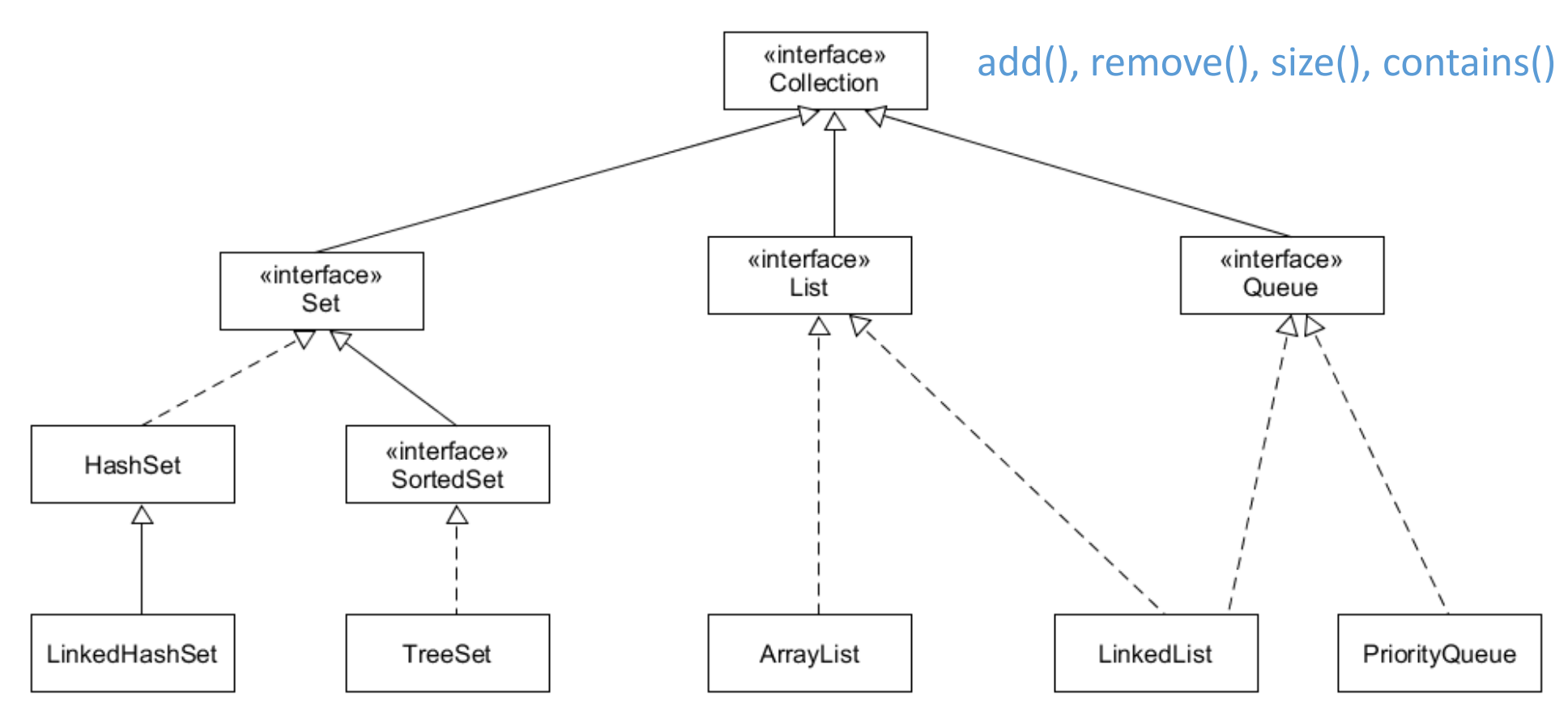
  - Utiliser une collection

```
List<Personne> equipe = new ArrayList<Personne>();
```

  - Utiliser un dictionnaire clé/valeur

```
Map<Integer, Personne> equipe = new HashMap<>();
```

Les collections



Les collections

Démonstration

Le dictionnaire clé/valeur

```java
//Déclarer et instancier le dictionnaire clé/valeur
Map<Integer, Employe> employes = new HashMap<Integer, Employe>();

//Ajouter les éléments
employes.put(emp1.getNoEmploye(), emp1);
employes.put(emp2.getNoEmploye(), emp2);
employes.put(emp3.getNoEmploye(), emp3);

//Accéder aux éléments par leur clé
Employe emp = employes.get(1);  // Employé no 1
```

La généricité

- Extension au typage fort
- Détecter les erreurs à la compilation
- Pouvoir passer un type en paramètre
- Syntaxe : utilisation du « diamant » <Type1[,Type2, …]>

Définir et utiliser la généricité

Type paramètre

```java
public class MonGenerique<T> {

    private T t;

    public void add(T t){
        this.t = t;
    }

    public T get(){
        return t;
    }

}
```

Type argument

```java
public static void main(String[] args) {
    MonGenerique<String> mc = new MonGenerique<>();
    mc.add("element");
    String s = mc.get();
}
```

La généricité

Démonstration

La généricité : appliquer des contraintes

MaClasse<T extends Vehicule>

MaClasse<T extends Vehicule & Comparable & Cloneable>

Paire<X extends Vehicule, Y extends Comparable>

~~MaClasse<T extends Personne & Vehicule>~~

~~MaClasse<T extends List & Vehicule>~~

JAVA – Notions complémentaires

Gestion d'un annuaire

TP