Ecole Supérieure
d'Informatique et du Numérique
COLLEGE OF ENGINEERING & ARCHITECTURE

Chapitre 6: Liste & Exceptions

M. NAJIB

Objectifs

L’objectif de ce chapitre est de présenter l’essentiel pour maitriser la création et la manipulation de la structure de données de type LINKEDLIST en JAVA.

À la fin de ce chapitre, vous êtes sensé acquérir les connaissances suivantes:

- Création des « Linkedlists »
- Ajouter, rechercher, modifier, supprimer les éléments d’une linkedlist
- Utilisation des exceptions par défaut
- Création des classes

Les collections d’objets en JAVA

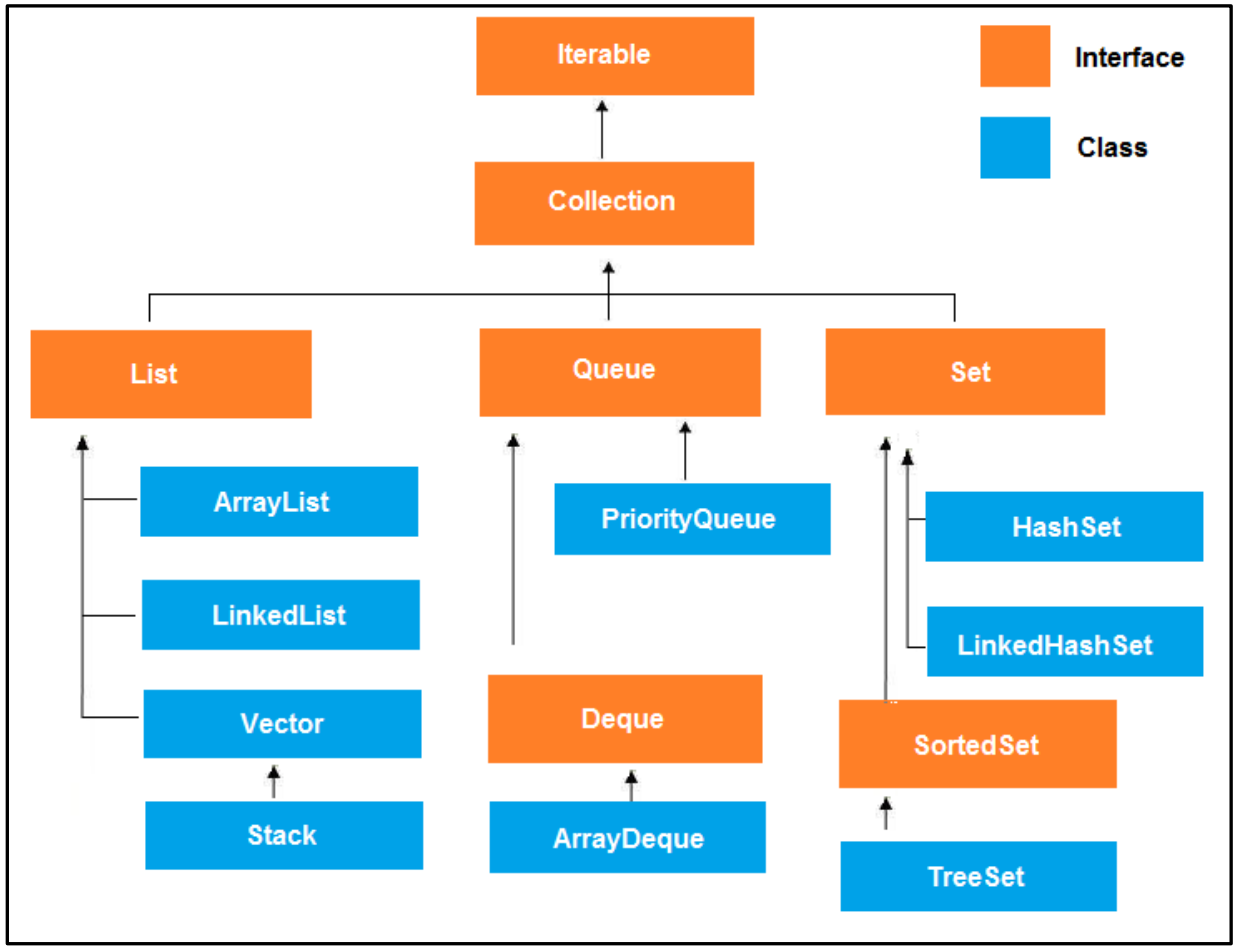
JAVA propose une bibliothèque riche et simple pour l’utilisation des structures de données.

Bibliothèque **java.util** propose les principales structures de données:

- Les vecteurs dynamiques (ArrayList et Vector)
- Les ensembles (TreeSet et HashSet)
- Les listes chaînées (LinkedList)
- Les queues (PriorityQueue)

Les collections d’objets en JAVA

Les collections en JAVA sont organisées de la manière suivante:



Les linkedlist

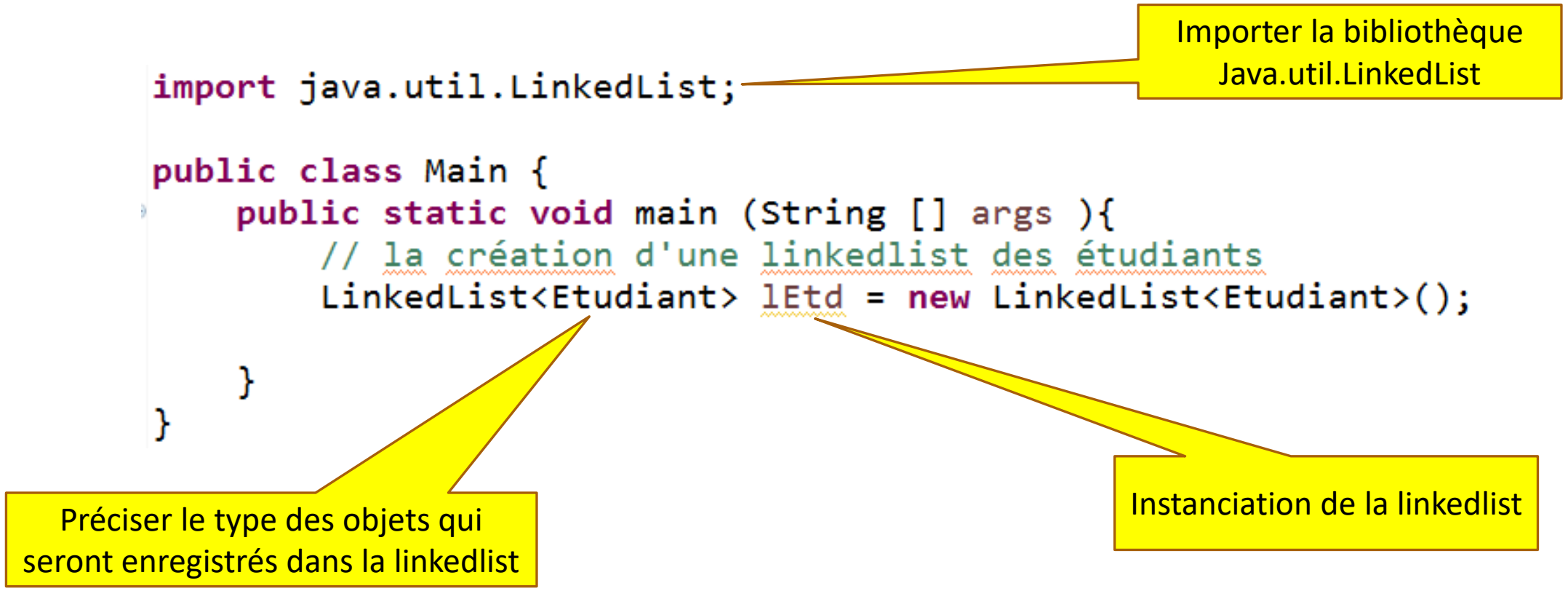
- Les listes chaînées en JAVA (Linkedlist en anglais) permettent la création des tableaux extensibles à volonté. Il s'agit des listes doublements chaînées.
- L'ajout d'un nouvel élément et la gestion de la mémoire est assurée automatiquement par la machine virtuelle.
- Les informations pour la déclaration des éléments qui précédent et succèdent un élément donné sont gérées automatiquement.

Point fort

Toutes les fonctionnalités des listes chaînées sans la gestion des pointeurs ☺

Les linkedlist

La création des Linkedlist se fait de la manière suivante:



Les linkedlist

La class Etudiant.java

```java
import java.util.LinkedList;

public class Etudiant {
    public int CNE;
    public String nom, prenom;
    public LinkedList<Integer> listeNote;

    public Etudiant(int cne, String nom, String prenom,
                    LinkedList<Integer> listeNote){
        this.CNE = cne;
        this.nom = nom;
        this.prenom = prenom;
        this.listeNote = listeNote;
    }
}
```

La déclaration de la liste des notes d'un étudiant

Les linkedlist - ajouter un objet

Pour ajouter un élément à une linkedList il faut utiliser la méthode **add(objet)**

Exemple:

```
Etudiant etd1 = new Etudiant(2534657, "NAJIB", "Mehdi", null );
Etudiant etd2 = new Etudiant(2534611, "Madhoun", "Soumia", null );
Etudiant etd3 = new Etudiant(2530000, "EDDAHMOUNI ", "HAMZA", null );

lEtd.add(etd1);
lEtd.add(etd2);
lEtd.add(etd3);
```

Utilisation de la méthode **add(objetEtd)**

Les linkedlist - ajouter un objet

Il existe d'autres variantes de la méthode add pour :

- **addFirst**(): ajouter un objet à la première position de la liste
- **addLast**(): ajouter un objet à la dernière position de la liste
- **addAll**(): ajouter une linkedList à une autre linkedList

Exemple:

Les linkedlist - ajouter un objet

Exemple

```
// la création d'une linkedlist des étudiants
LinkedList<Etudiant> lEtd = new LinkedList<Etudiant>();
LinkedList<Etudiant> lEtd2 = new LinkedList<Etudiant>();

Etudiant etd1 = new Etudiant(2534657, "NAJIB", "Mehdi", null );
Etudiant etd2 = new Etudiant(2534611, "Madhoun", "Soumia", null );
Etudiant etd3 = new Etudiant(2530000, "EDDAHMOUNI ", "HAMZA", null );

lEtd.addFirst(etd1);
lEtd.addFirst(etd2);
lEtd.addLast(etd3);
// remplir la deuxième liste
lEtd2.add(etd1);
lEtd2.add(etd3);
// ajouter la liste
lEtd.addAll(lEtd2);
```

Utilisation de la méthode **add(objetEtd)**

Remplissage de la deuxième liste

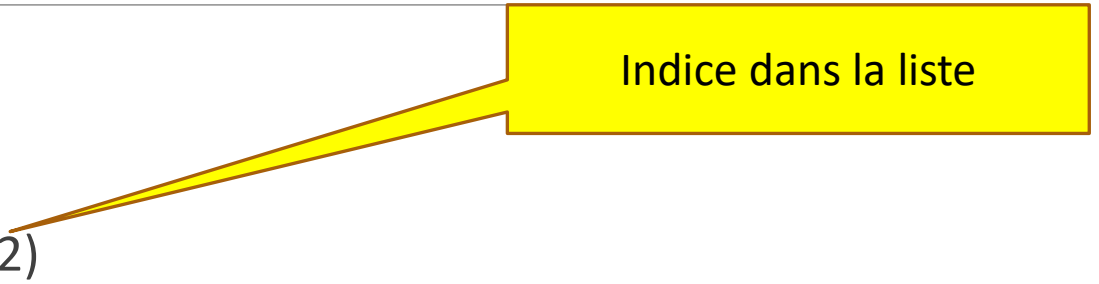
Ajout de la deuxième liste dans la 1ère liste

Les linkedList – supprimer un objet

On peut supprimer un objet de la liste:

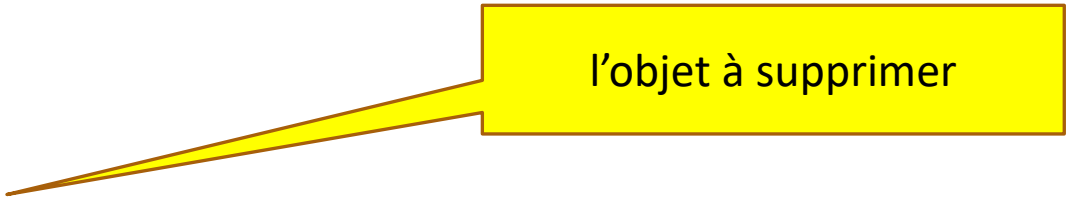
La suppression par indice :

lEtd.**remove**(2)



La suppression par recherche de l'objet:

lEtd.**remove**(etd1)



Les linkedlist – parcours

Il existe deux façons pour parcourir une linkedList

- **1ère solution**: utilisation d'une boucle et de la méthode **get(i)**

```
for(int i = 0; i < lEtd.size(); i++){

    System.out.println(lEtd.get(i));
}
```

La taille de la liste

Récupération du ième objet

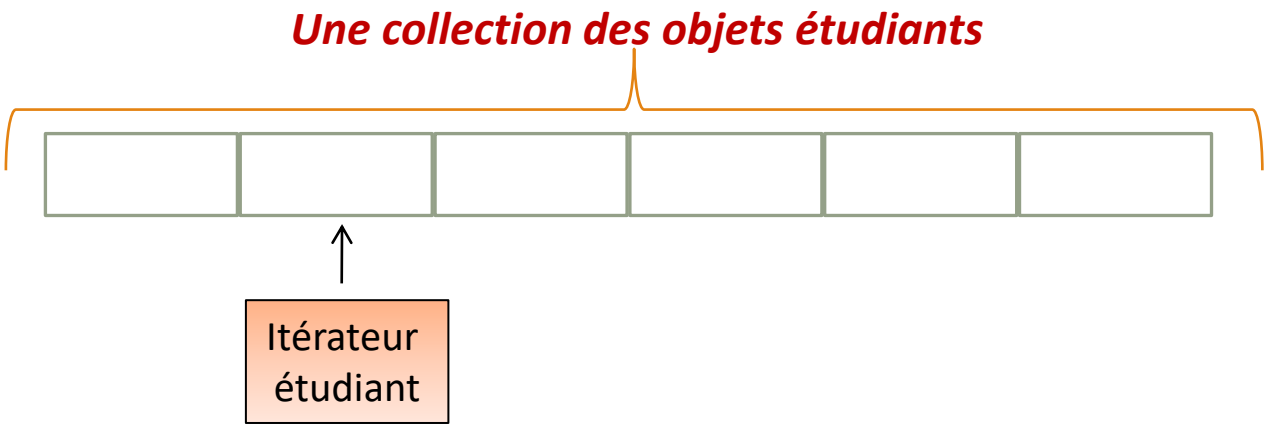
- **Résultat:**

```
<terminated> Main (3) [Java Application] C:\Program Files\Java\jre1.8.0
[Etudiant: 2534611 - Madhoun - Soumia ]
[Etudiant: 2534657 - NAJIB - Mehdi ]
[Etudiant: 2530000 - EDDAHMOUNI  - HAMZA ]
[Etudiant: 2534657 - NAJIB - Mehdi ]
[Etudiant: 2530000 - EDDAHMOUNI  - HAMZA ]
```

Les linkedlist – parcours (Iterator)

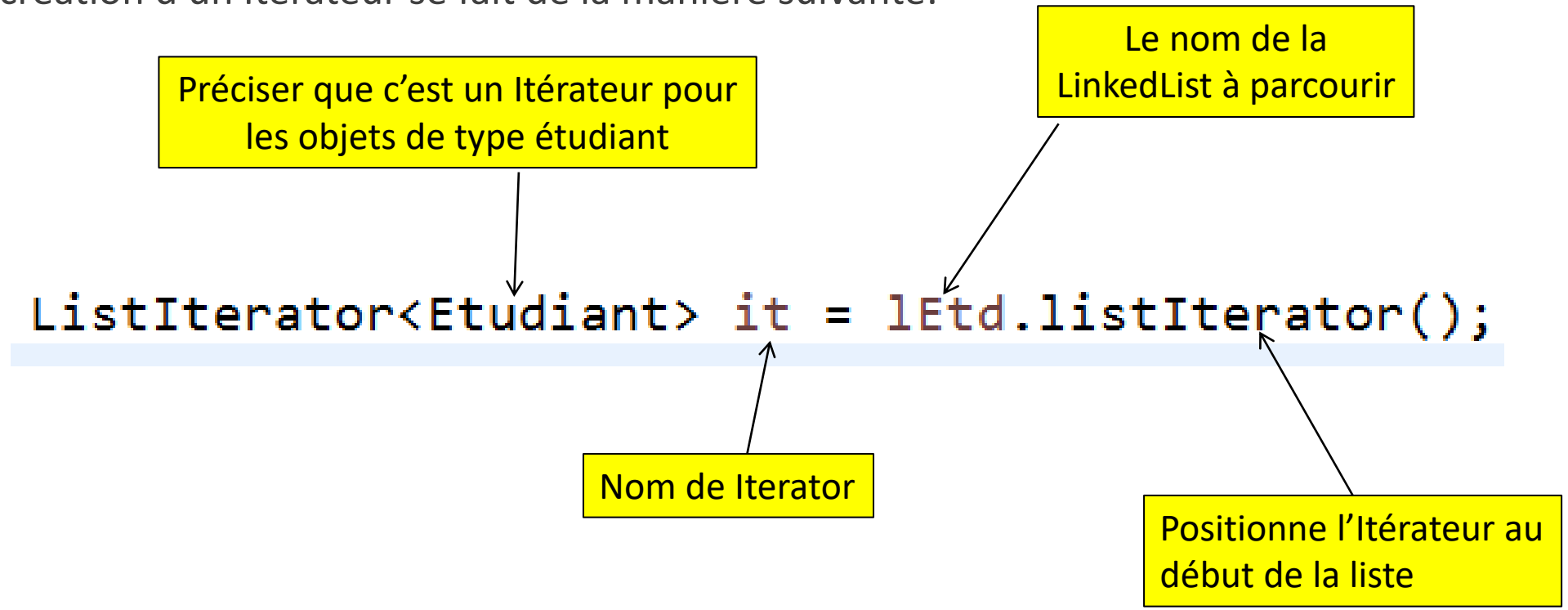
La deuxième solution se base sur l'utilisation des itérateurs.

Un Itérateur est un élément qui permet de parcourir une collection d'objets.



Les linkedlist – parcours (iterator)

La création d'un Itérateur se fait de la manière suivante:



Les linkedlist – parcours (iterator)

Un Itérateur propose un ensemble de méthode pour parcourir une collection

- **Iterateur.next():** retourne l'objet pointé par l'itérateur et positionne l'itérateur sur l'objet suivant
- **Iterateur.previous():** retourne l'objet pointé par l'itérateur et positionne l'itéreur sur l'objet précedent
- **Iterateur.hasNext():** teste si un objet existe après l'objet dans la collection
- **Iterateur.hasprevious():** teste si un objet existe avant l'objet pointé par l'itérateur

Les linkedlist – parcours (iterator)

Exemple : parcours en avant d'une LinkedList

```
ListIterator<Etudiant> it = lEtd.listIterator();

while(it.hasNext()){
    System.out.println(it.next());
}
```

Teste s'il y a d'autres éléments dans la liste

Objet étudiant

Résultat

```
<terminated> Main (3) [Java Application] C:\Program Files\Java\jre1.8.0
[Etudiant: 2534611 - Madhoun - Soumia ]
[Etudiant: 2534657 - NAJIB - Mehdi ]
[Etudiant: 2530000 - EDDAHMOUNI  - HAMZA ]
[Etudiant: 2534657 - NAJIB - Mehdi ]
[Etudiant: 2530000 - EDDAHMOUNI  - HAMZA ]
```

Les linkedlist – parcours (iterator)

Exemple : parcours en arrière d'une LinkedList

```
it = lEtd.listIterator(lEtd.size());

while(it.hasPrevious()){
    System.out.println(it.previous());
}
```

Se positionner à la fin de la liste

Teste si un objet existe l'objet courant

Résultat

```
[Etudiant: 2530000 - EDDAHMOUNI  - HAMZA ]
[Etudiant: 2534657 - NAJIB - Mehdi ]
[Etudiant: 2530000 - EDDAHMOUNI  - HAMZA ]
[Etudiant: 2534657 - NAJIB - Mehdi ]
[Etudiant: 2534611 - Madhoun - Soumia ]
```

Méthode compareTo

La méthode compareTo permet de comparer deux objets:

```
public int compareTo(Etudiant etd){
    if (this.moyenne == etd.moyenne)
        return 0;
    else if (this.moyenne > etd.moyenne)
        return -1;
    else
        return 1;
}
```

Méthode qui compare deux objets de type étudiant en se basant sur l'attribut moyenne

Méthode compareTo

On peut utiliser la méthode compareTo pour assurer le tri de la collection des étudiants

Exemple: Tri ascendant

```
// le tri d'une collection
Etudiant tempor = new Etudiant(0, "", "", 0);
// boucle
for (int i = 0; i < lEtd.size(); i++){
    for(int j = 0; j < lEtd.size(); j++){
        if(lEtd.get(i).compareTo(lEtd.get(j)) == 1){
            tempor = lEtd.get(i);
            // remplacer l'élément
            lEtd.set(i, lEtd.get(j));
            lEtd.set(j,tempor);
        }
    }
}
```

Méthode compareTo

Le résultat du Tri: une liste des étudiants triée dans l'ordre ascendant des moyennes

```
[Etudiant: 2530000 - EDDAHMOUNI  - HAMZA - 18.0]
[Etudiant: 2530000 - EDDAHMOUNI  - HAMZA - 18.0]
[Etudiant: 2534611 - Madhoun - Soumia - 19.0]
[Etudiant: 2534657 - NAJIB - Mehdi - 20.0]
[Etudiant: 2534657 - NAJIB - Mehdi - 20.0]
```

Interface comparator

```java
import java.util.Comparator;

public class CompareP implements Comparator<Personne>{
    @Override
    public int compare(Personne o1, Personne o2) {
        return (int) ( o1.note - o2.note);
    }
}
```

Méthode sort

```
public class Test {

    public static void main(String[] args) {

        LinkedList<Personne> listeP = new LinkedList<Personne>();

        listeP.add(new Personne(1, "etd1", 11));
        listeP.add(new Personne(2, "etd2", 15));
        listeP.add(new Personne(3, "etd3", 10));


        listeP.sort(new CompareP());

        System.out.println(""  + Arrays.toString(listeP.toArray()));


    }

}
```

Collections sort

Définir le comparateur durant le tri

```
Collections.sort(listeP,new Comparator<Personne>() {

    @Override
    public int compare(Personne o1, Personne o2) {
        return (int) ( o1.note - o2.note);

    }
});

System.out.println(""  + Arrays.toString(listeP.toArray()));
```

Exercice d’application

- Un « **EWallet** » est caractérisé par « idEtd » (auto-inc), solde (double), et une liste des opérations de paiement effectuées
- Un « **Paiement** » est caractérisé par un « idP », montant (double), description

La liste des Wallets est une liste static déclarée dans la classe principale

Question

1. Donnez le code de la méthode qui permettra de transférer du solde d’un porte-monnaie à un autre.
2. Suppression des redondances de la liste des Paiement et récupération du solde.

Part II. Exception

Exception de base

Personnalisation des exceptions

Objectifs

Ce chapitre se focalise sur la composante principale pour la gestion des erreurs en JAVA qui est la gestion des exceptions.

- Utilisation des exceptions par défaut
- Création des classes pour une gestion personnalisée des exceptions

Les exceptions

- Partant du principe qu'aucun programme (de taille importante) ne peut être parfait (gestion de tous les bugs possible).
- Les circonstances exceptionnelles peuvent engendrer des situations de la suspension de l'exécution du programme.

- **Exemple:**

1- De la division par zéro généré par la saisie de l'utilisateur

2- Lecture d'un fichier supprimé par un autre programme

Les exceptions

- La gestion de toutes les situations susceptibles d'engendrer des problèmes est une tâche fastidieuse.
- Impacter la lisibilité du code par l'introduction de plusieurs tests.

- Pour pallier ce problème JAVA propose l'utilisation des exceptions qui permettent de:

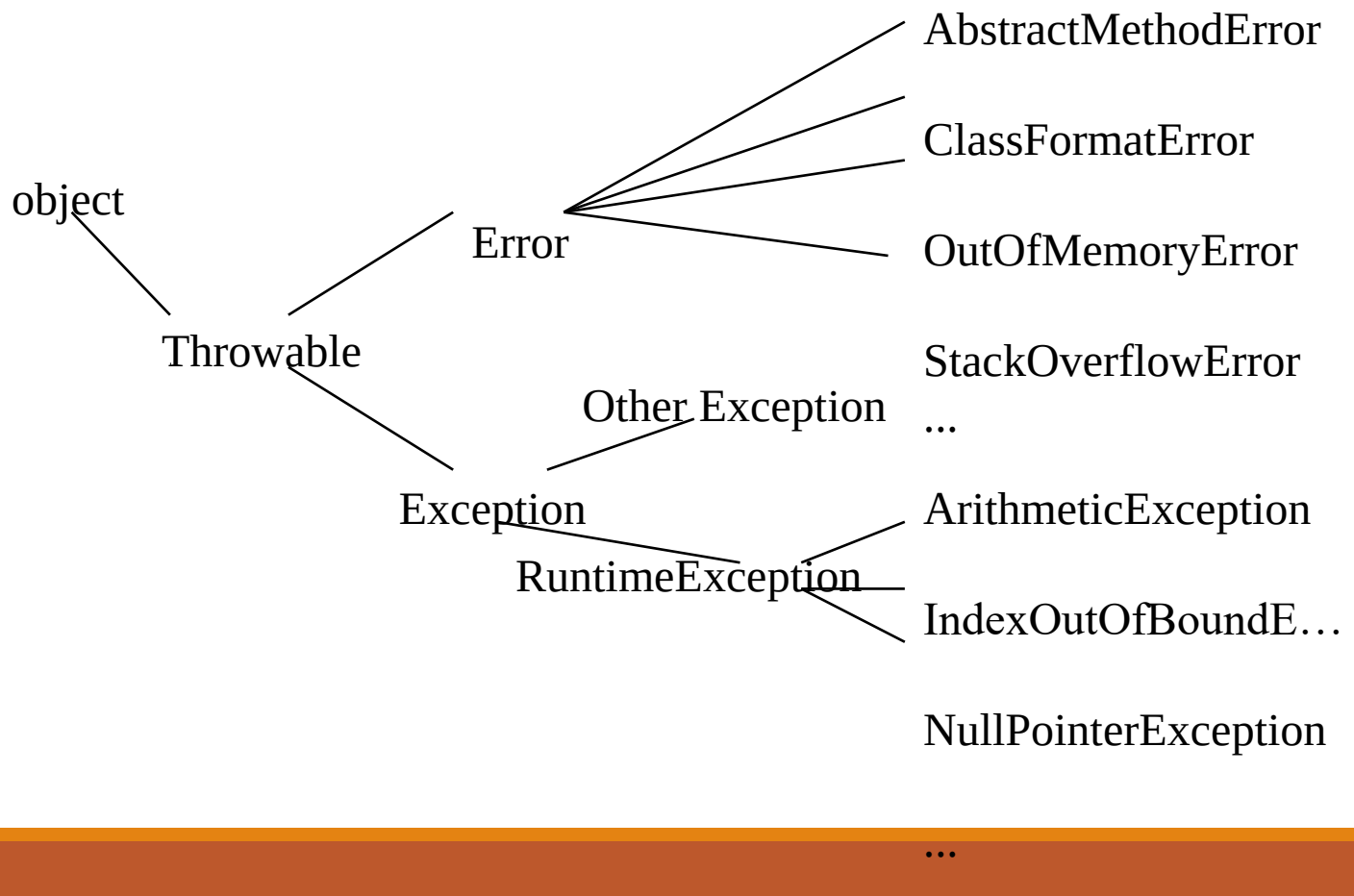
- **Dissocier la détection d'une anomalie de son traitement**
- **Séparer la gestion des anomalies du reste du code**

Types d’exception

Les exceptions prédéfinies se subdivisent en deux grandes catégories :

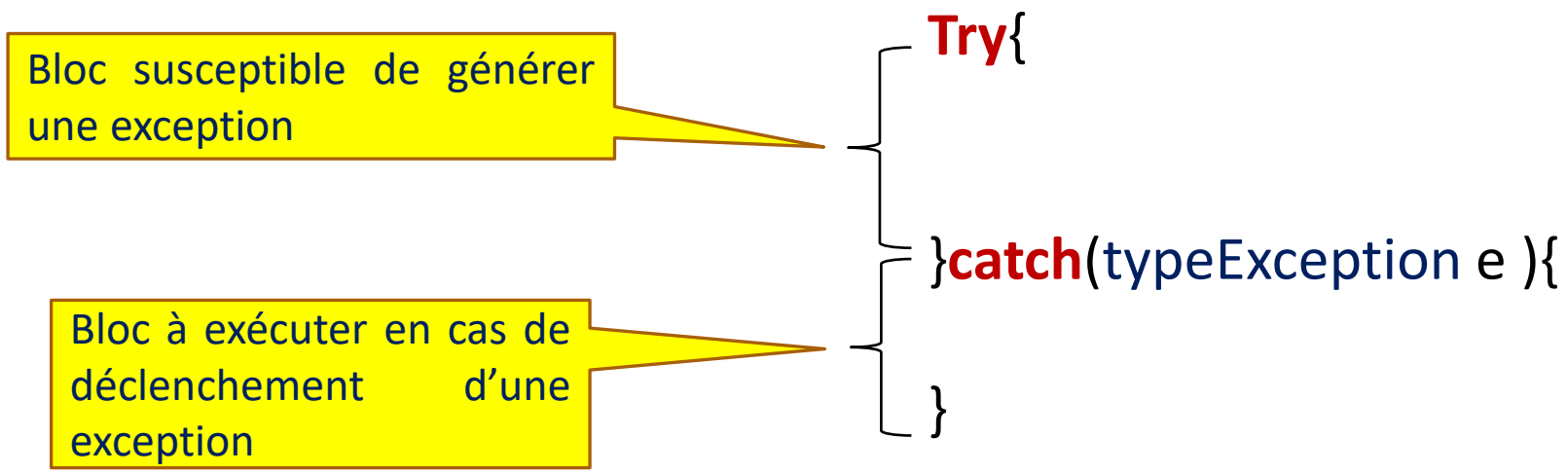
- les **Error** : Exceptions réservées aux erreurs systèmes. Elles peuvent être attrapées et traitées uniquement. Elles ne peuvent être levées
- les **RuntimeException**: Elles peuvent être attrapées, traitées et levées. Mais il est fortement conseillée de ne pas en définir

Types d’exception



Les exceptions

La déclaration des exceptions en JAVA se fait de la manière suivante:



Les exceptions

- L'exemple suivant montre la syntaxe de base pour la création d'une exception
- La classe exception

```
try{
    int a   = 1;
    int b   = 0;
    float c = a/b;
}catch(Exception e){
    System.out.println("execution des exceptions");
    e.printStackTrace();
}
```

Bloc susceptible de générer une exception

Bloc à exécuter après le déclenchement d'une exception

Les exceptions

- En JAVA on peut créer des exceptions spécifiques pour les cas à programmer
- Pour ce faire, il faut créer une classe qui hérite de la classe Exception (qui existe en JAVA)

```
public class PointException extends Exception  {

}
```

La classe exception

Les exceptions

La classe point Exception

Il faut ajouter le bloc **throws** pour spécifier
Que le constructeur déclenche une exception

```
public class Point {
    int x, y;
    public Point(int x, int y)throws PointException{
        if(x < 0 || y < 0) throw new PointException();
            this.x = x;
            this.y = y;
    }
    // throws - throw
    public String toString(){
        return "Point: " + this.x + " - " + this.y;
    }
}
```

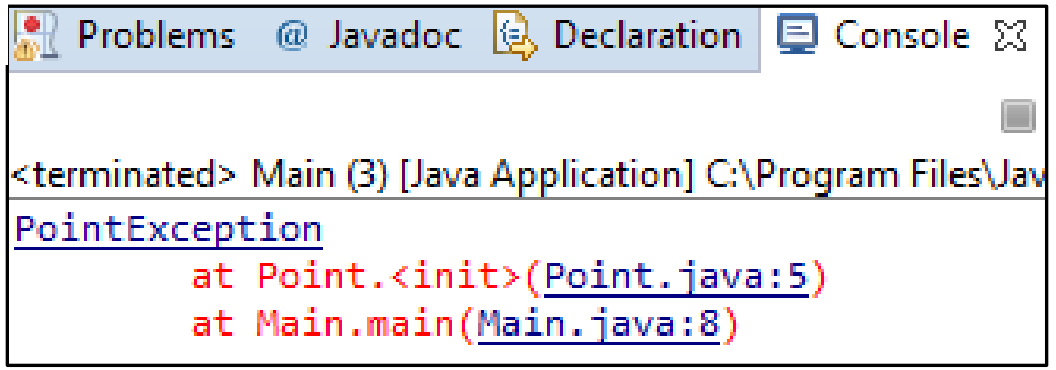
Spécification de la condition à vérifier
pour déclarer une exception

Les exceptions

Le code Main pour tester notre class d'exception

```
try{
     Point p = new Point(-1,-2);
     System.out.println(p);
}catch(PointException e){
     e.printStackTrace();
}
```

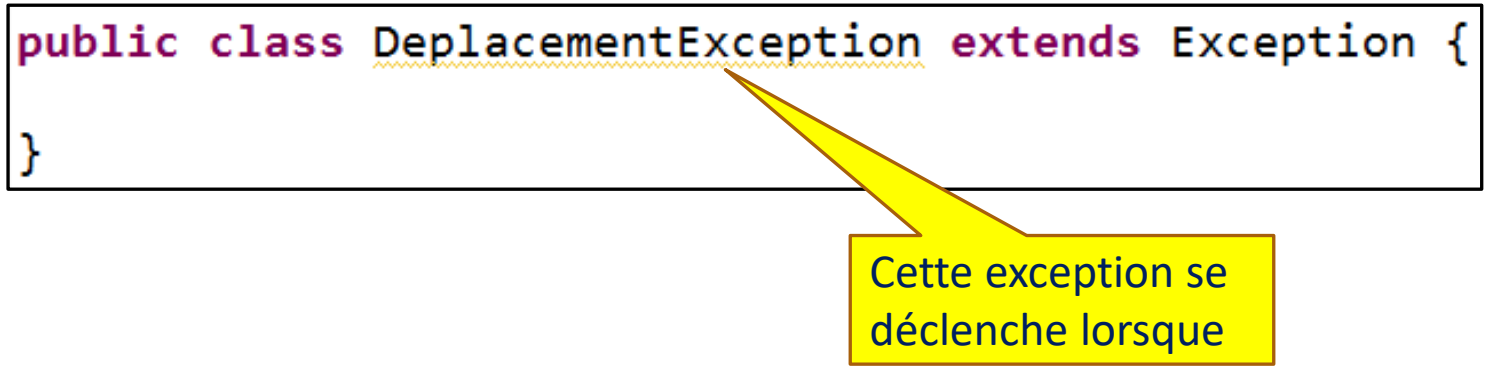
Résultat



Les exceptions

JAVA permet à une méthode de déclencher plusieurs types d’ exception en même temps

Pour tester ce cas nous allons créer une deuxième classe exception.



Les exceptions

La gestion du déclenchement des deux exceptions

```
try{
    Point p = new Point(-1,-2);// excepttion du constructeur
    System.out.println(p);

    p.deplacer(-10, -10);// exception de deplacement
    System.out.println(p);

}catch(PointException | DeplacementException e){
    e.printStackTrace();
}
```

1ère exception: instanciation

2ème exception: déplacement

Le déclenchement de la première exception bloque l'exécution de la deuxième exception.

Les exceptions

Une deuxième manière d'implémentation

```
try{
    Point p = new Point(-1,-2);// excepttion du constructeur
    System.out.println(p);

    p.deplacer(-10, -10);// exception de deplacement
    System.out.println(p);

}catch(PointException e){
    System.out.println("exception constrcution");
    e.printStackTrace();
    System.exit(-1);

}catch(DeplacementException e){
    System.out.println("exception deplacement");
    e.printStackTrace();
    System.exit(-1);
}
```

```
<terminated> Main (3) [Java Application] C:\Program Files'
exception constrcution
PointException
        at Point.<init>(Point.java:5)
        at Main.main(Main.java:8)
```

Les exceptions

Personnalisation des exceptions:

```
public class PointException extends Exception  {
    public int a, b;
    public PointException(int a, int y ){
        this.a = a;
        this.b = y;
    }
}
```

Passage des paramètres aux constructeurs de la classe exception

Les exceptions

Passage des paramètres à la méthode de déclenchement des exceptions

```
public class Point {
    int x, y;
    public Point(int x, int y)throws PointException{
        if(x < 0 || y < 0) throw new PointException(x,y);
            this.x = x;
            this.y = y;
    }
```

Exemple du passage des attributs de la classe point dans la méthode qui déclenche l'exception

Les exceptions

Accès aux attributs

```
public static void main (String [] args ) {
    try{
        Point p = new Point(-1,-2);// excepttion du constructeur
        System.out.println(p);

        p.deplacer(-10, -10);// exception de deplacement
        System.out.println(p);

    }catch(PointException e){
        System.out.print("exception constrcution");
        System.out.println(" a= " +  e.a + " b= " + e.b);
        e.printStackTrace();
        System.exit(-1);
```

Récupération des paramètres

```
<terminated> Main (3) [Java Application] C:\Progra
exception constrcution a= -1 b= -2
PointException
        at Point.<init>(Point.java:5)
        at Main.main(Main.java:8)
```

Les exceptions – le bloc finally

Le bloc finally se déclenche

```
public static void main (String [] args ) {

    try{
        Point p = new Point(-1,-2);// excepttion du constructeur
        System.out.println(p);

        p.deplacer(-10, -10);// exception de deplacement
        System.out.println(p);

    }catch(PointException e){
        System.out.println("exception constrcution");
    }catch(DeplacementException e){
        System.out.println("exception deplacement");
    }finally{
        System.out.println("le bloc qui sera toujours déclencher ");
    }
```

Le bloc finally qui sera toujours exécuté après le déclenchement des exceptions

Exercice d’application

Proposez une solution pour déclencher une exception qui protège contre les erreurs de saisie de l'âge d’un étudiant qui souhaite faire son inscription à l’ESIN.

- Erreur de saisie
- Age de l’étudiant est inférieur à 17 ans

Class étudiant (nom Etudiant, age)

Merci de votre attention