Java™

Master science de donnée et aide à la décision FSTG

Réaliser par
ilhame souda
Rachida oubouali
Doha ayougil
Hasna Amrani

Plan

Introduction general

Géniricité , Collections ,Threads

Concepts de base

Exceptions,Chaine de caracteres

Classes,Objets,Tableaux

Heritage,package,Classes imbriquées

Interfaces,Classes abstraites

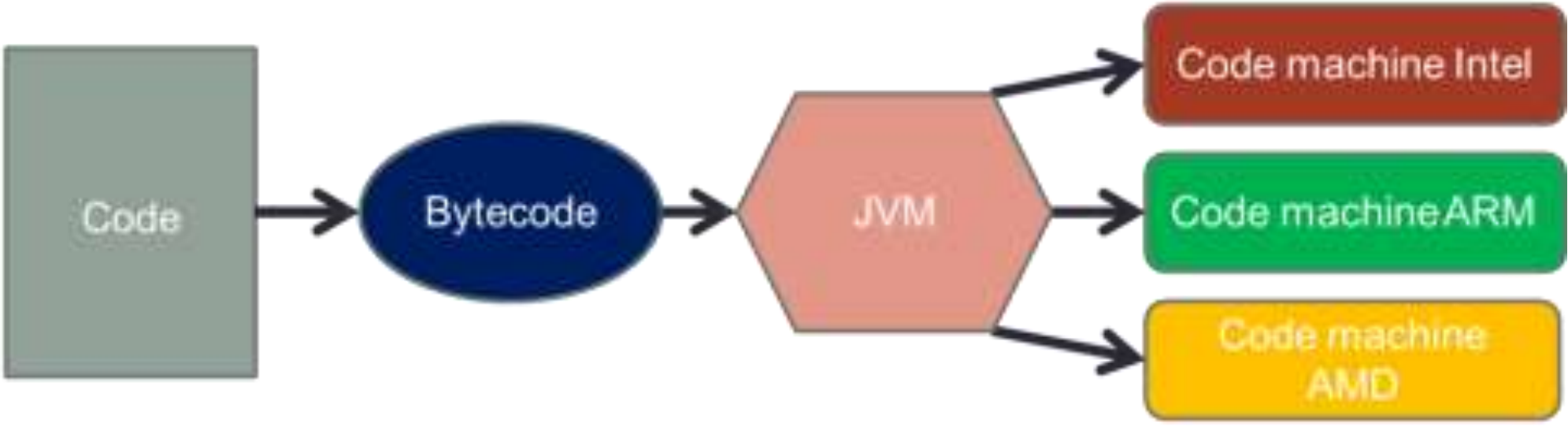
INTRODUCTION À

Java - Historique

- Développé par Sun Microsystems à partir de 1991 par
  - Le canadien James Gosling
  - Et toute une équipe
- La première version est disponible à partir de 1995
- La dernière version, Java SE 12, est disponible depuis mars 2019
- Nous utiliserons Java SE 8 pour le cours.

Compilation / Interprétation

Exécution d'un programme interprété(Java, C#, etc.)



Pour obtenir un programme :

- Pour du Java, on compile le code en *bytecode*. La machine virtuelle transforme le *bytecode* en code machine au fur et à mesure que le programme est exécuté.

```
javac PremierProg.java
java PremierProg
```

Approche OrientéeObjet

Le concept de programmation orienté objet vient de la volonté d'avoir une représentation très proche du monde réel. En effet, ce monde est constitué d'objets. Ils sont caractérisés par leur dimension, leur poids, leur couleur, etc.

Le concept de programmation orientée objet vient donc pour répondre à un certain nombre d'objectifs dont :

- Similarité avec le monde réel
- Similarité avec le langage naturel
- Prise en compte de l'évolution du besoin
- Prise en compte des systèmes existants

Principe/Avantages

Un objet est une abstraction d'un élément du monde réel. (Exemple :Voiture) possède des informations (=attributs=l'état de l'objet), par exemple Couleur, Matricule, modèle, etc., et se comporte suivant un ensemble d'opérations qui lui sont applicables: vitesse, accélérer, freiner, tourner, etc.

- Objet = attributs+ opérations

De plus, un ensemble d'attributs caractérisent l'état d'un objet, et l'on dispose d'un ensemble d'opérations (les méthodes) qui permettent d'agir sur le comportement de notreobjet.

Principe/Avantages

Les avantages de l'approche objet:

- Programmation orientée objet apporte l'indépendance entre les programmes, les données et les procédures parce que les programmes peuvent partager les mêmes objets sans avoir à se connaître comme avec le mécanisme d'import/export.

- La modélisation des objets de l'application : informatiquement un ensemble d'éléments d'une partie du monde réel en un ensemble d'entités informatiques. Ces entités informatiques sont appelées objets

Qu'est-ce qu'un objet?

En programmation orientée objet, chaque entité du monde réel doit être représentée justement par un objet.

A chaque instant, cette entité (objet) est caractérisée par:

- L'information qui ressort de la valeur des données (son état : attribut)
- Les opérations qui peuvent modifier son état ou interagir avec les autres objets (méthodes)
- L'identité qui caractérise l'existence propre d'un objet et permet de distinguer deux objets dont toutes les valeurs d'attributs sont identiques.

La notion de classe

Une Classe est un modèle informatique représentant une famille d'objets ayant :

✓ La même structure de données (même liste d'attributs)

✓ Les mêmes méthodes (mêmes comportements).

- La création d'un objet en tant qu'exemplaire concret (contenant des données) d'une classe s'appelle une **INSTANCIATION.**
- Chaque objet donne des valeurs aux attributs de la classe.

Classe = attributs + opérations + instanciations

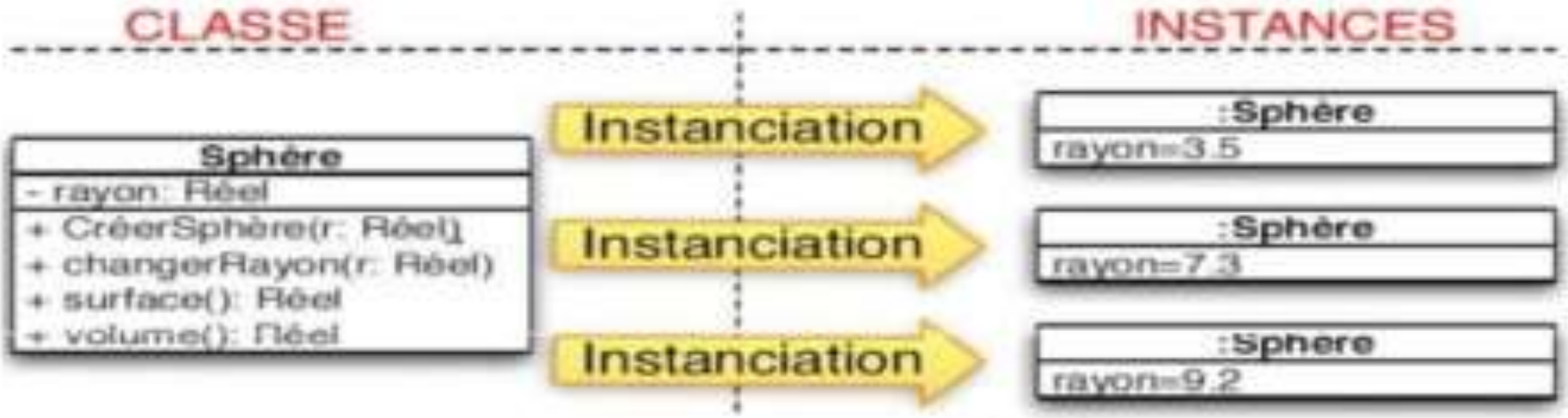
Les attributs (appelés aussi variables d'instances): Ils ont un nom , soit Un type primitif (Entier, Booléen,...) soit une classe (Rectangle, Personne, ...) et une visibilité

La notion de classe

Les opérations (méthodes): Elles sont les opérations applicables à un objet de la classe. On a **5 catégories d'opérations**:

Constructeurs (qui créent les objets), **Destructeurs** (qui détruisent les objets), **Sélecteurs** (qui renvoient tout ou une partie de l'état d'un objet), **Modificateurs** (qui changent tout ou partie de l'état d'un objet), **Itérateurs** (qui visitent l'état d'un objet ou le contenu d'une structure de données contenant des objets)

L'instanciation : L'objet possède une identité, qui permet de le distinguer des autres objets, indépendamment de son état.



La notion de classe

Visibilité:

Il existe trois niveaux de visibilité :

publique : Les fonctions de toutes les classes peuvent accéder aux données ou aux méthodes d'une classe définie avec le niveau de visibilité public. Il s'agit du plus bas niveau de protection de données.
protégée : l'accès aux données est réservé aux fonctions des classes héritières, c'est-à-dire par les fonctions membres de la classe ainsi que des classes dérivées.
privée : l'accès aux données est limité aux méthodes de la classe elle-même. Il s'agit du niveau de protection des données le plus élevé.

La notion de classe

Exemple d'une classe en java

Classe représentant un point de R2

Opération : translater le point d'un vecteur (tx, ty)

```
public class Point
{ private double x;
   private double y;
      public void translate(double x, double y){ this.x += x;
                                                  this.y += y; }
}
```

public class Point : définition d'une nouvelle classe Point

private double x : attributs de la classe avec mode d'accès private= accessible uniquement depuis cette classe; accès à l'intérieur de la classe : this.nomAttribut

public void translate(double tx, double ty) : méthode de la classe

L'analyse orienté objet

L'analyse orienté objet s'appuie sur les principes suivants :

Abstraction : décrire formellement les données et les traitements en ignorants les détails.

Encapsulation : masquer les données et les traitementsen définissant une interface.

Héritage : reprendre intégralement tout ce qui a déjà été fait et de pouvoir l'enrichir : spécialisation.

Polymorphisme : le terme polymorphisme issu du grec signifie la faculté
de prendre plusieurs formes.

Messages : le seul moyen de communication entre les objets et l'envoi de messages.

Notion de constructeur

constructeur = méthode particulière permettant de créer des instances de la classe

- objectif = initialiser les différents attributs
- définition d'un constructeur =
  - même nom que la classe
  - pas de type de retour
  - autant de paramètres que ce dont on a besoin

Il existe deux types de constructeur en Java:

-Constructeur sans argument : Un constructeur sans paramètre est appelé constructeur par défaut

Exemple: Cube OB = newCube();

-Constructeur paramétré: Un constructeur qui a desparamètres est appelé constructeur paramétré.

Exemple: Cube OB=new Cube(String nom, int age)

Avantages de la POO

- les données sont protégés et accessibles d'une manière bien définie
- la modification du code des méthodes d'une classe n'influence en aucun cas les programmes externes utilisant cette classe
- les programmeurs peuvent réutiliser des composants existants

Notion de Classes

- Java est un langage objet: tout appartient à une classe sauf les variables de type primitives.
- Une classe se compose en deux parties :
  - Le corps peut être divisé en 2 sections : la déclaration des données et des constantes et la définition des méthodes.
  - Les méthodes et les données sont pourvues d'attributs de visibilité qui gère leur accessibilité par les composants hors de la classe.
- Un attribut est une donnée d'une classe dont la déclaration respecte la syntaxe:

```
1 type_de_attribut nom_de_attribut = [valeur ] ;
```

Les constructeurs & les méthodes d'une classe

- Un constructeur est une méthode automatiquement appelée lors de la création d'une instance d'une classe.
- Si aucun constructeur n'est défini par le programmeur, java considère que la classe est munie d'un constructeur par défaut.

- Une fonction d'une classe est appelée méthode.
- En java une méthode peut retourner une valeur et avoir une liste de paramètres:
  - Une méthode peut avoir un nombre quelconque de paramètres.
  - Tous les paramètres sont passés par valeur.

```
1 type_de_retour nom_de_méthode (liste_des_paramètres){
2 // corps de la méthode}
```

Exemple de classe

```java
public class Livre {
private String titre, auteur; //Variables d'instance
private int nbPages;
// Constructeur
public Livre(String unTitre, String unAuteur){
titre = unTitre;
auteur = unAuteur;
}
//méthode accesseur
public String getAuteur() {
return auteur;
}
// méthode modificateur
public void setNbPages(int nb) {
nbPages = nb;
}
}
```

Notion d'objets

- La création des objets est réalisée par l'opérateur new.

```
1 NomDeClasse objet;
2 objet=new NomDeClasse (paramètres de constructeur);
```

```
1 String S=new String("ceci est une chaîne de caractères");
```

- Pour accéder à une classe il faut en déclarer une instance de classe ou objet.

Exemple : utilisation de la classe Livre

```
1 class TestLivre {
2 public static void main(String[] args) {
3 Livre livre = new Livre("Les Misérables", "Victor Hugo");
4 livre.setNbPages(450);
5 System.out.println(livre.getAuteur());
6 }
7 }
```

La méthode main

- Pour débuter une application on doit fournir un point d'entrée. Lorsqu'une classesert d'application elle doit fournir ce point d'entrée qui est une méthode statique portant le nom de «main».
- Ce point d'entrée doit respecter la syntaxe suivante :

```
1    public static void main( String [] args ){
2  // Corps du point d'entrée
3 }
```

- Où «args» correspond à la liste des arguments passé depuis la ligne de commande.

```
1 public class PremierProgrammeJava{
2 public static void main(String args[]){
3 System.out.println("Bonjour c'est mon premier programme en Java");
4 }
5 }
```

Modificateurs et visibilité

- les modificateurs permettant de contrôler l'accès à une classe, une méthode ou une variable : public, private et protected.
- Une donnée ou méthode private est inaccessible depuis l'extérieur de la classe où elle est définie même dan une classe dérivée.
- Une donnée ou méthode public est accessible depuis l'extérieur de la classe où elle est définie.
- Une donnée ou méthode protected est protégée de tout accès externe comme private sauf à partir des classes dérivées (accessible dans une classe fille).

- Le modificateur static : est utilisé pour définir des variables ou des méthodes de classe. Le mot-clé static indique quel'élément est stocké dans la classe.
- Le modificateur final indique que la « variable » est constante.

Tableaux à une seule dimension

- Un tableau est une structure de données contenant un groupe d'éléments tous du même type.

```
1 Type NomDuTableau[];
```

- Les tableaux sont comme les instances de classes, créés par l'intermédiaire de l'opérateur new tout en précisant la taille désirée :

```
1 NomDuTableau = new Type[Taille
```

Exemple:

```
1 int T[]=new int[10];
```

Tableaux à plusieursdimensions

- S'il s'agit d'un tableau àplusieursdimensionsonutilise la syntaxe suivante:

```
1 Type NomDuTableau[][]...;
```

- Dans le cas d'un tableau d'objets, la création du tableau ne signifie pas celle des éléments du tableau

Exemple:

```
1 String TS[]=new String[3];
2 TS[0]=new String("abc");
3 TS[1]=new String("defgh");
4 TS[2]=new String("ijk");
```

Les classes Abstraites

- Une classe abstraite est une classe qui ne permet pas d'instancier des objets, elle ne peut servir que de classe de base pour une dérivation.
- Dans une classe abstraite, on peut trouver classiquement des méthodes et des champs, dont héritera toute classe dérivée et on peut trouver des méthodes dites « abstraites » qui fournissent uniquement la signature et le type de retour

Syntaxe

```
1  abstract class A
2  {public void f() {……} //f est définie dans A
3  public abstract void g (int n) ; //g est une méthode abstraite elle
4  n'est pas définie dans A
5  }
```

Règles des classes abstraites

- Dès qu'une classe abstraite comporte une ou plusieurs méthodes abstraites, elle est abstraite, et ce même si l'on n'indique pas le mot clé « abstract » devant sa déclaration.
- Une méthode abstraite doit être obligatoirement déclarée « public », ce qui est logique puisque sa vacation est d'être redéfinie dans une classe dérivée.
- Les noms d'arguments muets doivent figurer dans la définition d'une méthode abstraite
- Une classe dérivée d'une classe abstraite n'est pas obligée de redéfinir toutes les méthodes abstraites, elle peut ne redéfinir aucune, mais elle reste abstraite tant qu'il y a encore des méthodes abstraites non implémentées.
- Une classe dérivée d'une classe non abstraite peut être déclarée abstraite

EXEMPLE

```
1  abstract class graphique {
2  private int x, y;
3  graphique ( int x, int y) { this.x = x ; this.y = y ;}
4  void affiche () {
5  System.out.println (« Le centre de l'objet se trouve dans : » + x + « et
6  » + y) ;}
7
```

interface

Définition

Une interface définit un comportement (d'une classe) qui doit être implémenté par une classe, sans implémenter ce comportement. C'est un ensemble de méthodes abstraites, et de constantes. Certaines interfaces (*Cloneable*, *Serializable*, …) sont dites interfaces de «balisage» : elle n'imposent pas la définition d'une méthode, mais indiquent que les objets qui les implémentent ont certaines propriétés.

Les différences entre les interfaces et les classes abstraites :

- ✓ Une interface n'implémente aucune méthode.
- ✓ Une classe, ou une classe abstraite peut implémenter plusieurs interfaces, mais n'a qu'une super classe, alors qu'une interface peut dériver de plusieurs autres interfaces.
- ✓ Des classes non liées hiérarchiquement peuvent implémenter la même interface

L'interface *Comparable*

- L'interface *Comparable*, nécessite l'implémentation de la méthode *compareTo*

  Exemple: ordonner les employés d'une entreprise par ordre croissant des salaires :

```
1  public class Entreprise {
2      Employe [] lEntreprise;
3      public Entreprise(Employe [] e) {
4          lEntreprise = new Employe[e.length];
5          for (int i = 0; i<e.length; ++i)
6              lEntreprise[i] = e[i];
7      }
8      public void lister(){
9          for (int i = 0; i<lEntreprise.length; ++i)
10             System.out.println(lEntreprise[i]);
11     }
```

L'interface *Comparable*

```
1  class Employe implements Comparable{
2      . . .
3      public int compareTo( Object o ){
4          Employe e = (Employe)o; // lève une exception
5                                  // ClassCastException si o
6                                  // n'est pas de la classe Employe
7          return nom.compareTo(e.nom);// par ordre alphabétique
8          return (int)(salaire-e.salaire);// par salaire croissant
9      }
10 }
```

L'interface *Serializable*

- La *sérialisation* d'un objet est le processus de sauvegarde d'un objet complet sur fichier, d'où il pourra être restauré à tout moment. Le processus inverse de la sauvegarde (restauration) est connu sous le nom de *désérialisation*.
- Tous les attributs (qui ne sont pas spécifiés *transient*) d'une classe implémentant l'interface *Serializable* sont sauvés et restaurés de façon simple en utilisant les classes *ObjectOutputStream* et *ObjectInputStream*.

EX:

```
1  public class MaClasse implements Serializable{
2      String nom;
3      int entier;
4      MaClasse premier ,suivant;
5      public MaClasse(String n, int e, MaClasse s, MaClasse p) {
6          super() ;
7          nom = n;
8          entier = e;
9          suivant = s;
10         premier = p ;
11     }
12
13     public MaClasse(String n, int e) {
14         this(n, e, null, null);
15         premier = this;
16     }
17 }
```

L'interface *Iterator*

- L'interface *Iterator* nécessite l'implémentation des fonctions *hasNext()*, *next()*, *remove()*

```
1  interface Iterator {
2      boolean hasNext();
3      Object next();
4      void remove();
5  }
```

L'interface ListIterator

- L'interface *ListIterator* est dérivée de *Iterator*, et ajoute des fonctionnalités de parcours dans le sens inverse, de calcul d'indice, et d'ajout et de modification.

`boolean hasPrevious()`	Retourne vrai si l'élément courant à un élément le précédant
`Object previous()`	Retourne l'élément précédant.
`int nextIndex()`	Retourne l'indice de l'élément qui serait retourné par un appel de *next*
`int previousIndex()`	Retourne l'indice de l'élément qui serait retourné par un appel de *previous*
`void add(Object o)`	Ajoute un élément dans la liste (opération optionnelle)
`void set(Object o)`	Remplace le dernier élément retourné par *next* ou *previous* par *o* (opération optionnelle)

L'héritage en Java

- Pour réaliser une nouvelle classe B qui hérite d'une classe A existante, la syntaxe utilisée est la suivante :

```
1 class B extends A{
2 ...
3 }
```

- Un objet de la classe B peut maintenant appeler les méthodes publiques de B et de A.
- Si une méthode est définie à la fois dans B et dans A (redéfinition), alors celle invoquée par un objet de la classe B est la méthode redéfinie dans la classe B.

Exemple d'héritage

```
1 class Personne{
2 String nom ;
3 int nombreEnfants = 0 ;
4Personne(String n){
5 nom= n ; }
6 }
7 class Salarie extends Personne{
8 int salaire ;
9Salarie (String n, int s ) {
10 super (n) ;
11 salaire = s ; }
12 int prime (){
13float taux = nombreEnfants * 5 / 100 ; 14
return salaire * taux ;}
15 }
```

L'attribut caché this

- Chaque instance de classe comporte un attribut appelé this qui désigne l'instance courante de la classe.
- Cette attribut peut être utile pour retourner une référencevers l'instance ou pour lever certaines ambiguïtés sur les identifiants.

Exemple

```
1 public float longueur;
2 public void fixeLongueur(float longueur){
3 this.longueur = longueur;
4 }
```

Mot clé super

- Le mot clé super désigne la classe mère. Il peut être utilisé dans deux situations différentes :
  1. Pour appeler un constructeur de la classe mère : il est alors utilisé comme première instruction du constructeur de la classe fille :

Syntaxe

```
1 super (paramètres du constructeur de la classe mère)
```

  2. Pour appeler une méthode de la classe mère, si celle-ci est redéfinie dans la classe fille :

Syntaxe

```
1 super.nomDeLaMethode (paramètres );
```

Exemple d'appel d'un constructeur de la classe mère

Classe mère

```
1 class A{
2private string Name; 3
A(String S){ Name=S;}
4 String getName(){ return Name;}
5 }
```

Classe fille

```
1 class B extends A{
2 B(String Name){ super(Nom);}
3public static void main(String [] args){ 4
B objet = new B("objet B");
5 System.out.println(B.getName());}}
```

Exemple d'appel d'une méthode de la classe mère

Classe mère

```
1 class  A{
2 void p(){
3 System.out.println("Méthode  p()  de la  classe A " );}
4 } }
```

Classe fille

```
1 public class B extends A{
2 void  p(){
3System.out.println("Méthode p() de la classe B " );  4
super.p();}
5 public static void main(String [] args){
6 B objet = new B();
7 objet.p();}}
```

Les packages en Java

- Un package contient généralement plusieurs classes, qui sont définies généralement par plusieurs fichiers différents.
- Chaque paquetage porte un nom.
  - Ce nom est soit un simple identificateur ou une suite d'identificateurs séparés par des points.
- L'instruction permettant de nommer un paquetage doit figurer au début du fichier source (.java).
- On définit un package par le mot réservé package, suivi du nom de ce package, et d'un point-virgule.

```
1 package MonPackage;
2class ClasseA { ... int uneMethodeDeA(); ...} 3
class ClasseB { ... }
4 class  ClasseC { ... }
```

Importation de package

- Le langage java fournit une facilité d'écriture par l'emploi de la notion d'importation de package.
- En utilisant le mot clef import, on importe les définitions un ou plusieurs éléments d'un package.

Exemple

```
1 package Exemple;
2 class toto{ ...}
```

```
1 package Exemple;
2 class titi extends toto{...}
```

```
1 import Exemple.*;
2 class tutu extends titi{...}
```

Les aspects objets du langage

- Le langage java est un langage objet qui propose une classe de base appelée : java.lang.Object.
- Toutes les classes java héritent implicitement de java.lang.Object.
- Pour connaître le type réel d'un objet on utilise l'opérateur instanceof.

```
1    if ( objet instanceof maClasse )
2    // Bloc d'instructions
```

L'héritage et la classe Object

- L'héritage est un concept qui est toujours utilisé en Java.
  - Ainsi àchaquefois que l'on définisse une nouvelle classe, celle-ci ce dérive automatiquementde la classe Object mère de toutes les classes.

```
1 class Test{
2 Test(){
3 String S=getClasse().getName();
4 System.out.println("Je suis un nouvel objet de la
  classe :" + S);
5 }
6 public static void main(String args[]){
7 newTest();
8 }}
```

- A l'exécution le programme affiche le message: Je suis un nouvel objet de la classe : Test

Les méthodes de la classe Object

- getClasse() étant une méthode de la classe Object retournant un objet de type Class qui représente la classe de l'objet appelant.
- La méthode getName() de la classe Class permet d'obtenir le nom de la classe sous frome d'une chaîne de caractères.
- protected Object clone() : créer une copie de l'objet appelant.
- boolean equals(Object obj) : teste si l'objet obj est égale à l'objet appelant. La fonction retourne true si les objets ont la même référence.

```
1    class1 v1= new class1 ();
2    class1 v2=v1;
3    if(v1.equals(v2))...
```

  - La méthode equals peut être redéfinie pour faire une comparaison sur le contenu et non la référence.

Les méthodes de la classe Object

- protected void finalize() : cette méthode est appelée par le Garbage Collector lorsqu'il veut éliminer un objet de la mémoire.
  - Si l'on veut réaliser un traitement particulier il suffit d'appeler et de redéfinir cette méthode avec le comportement adéquat.
- Class getClass() : retourne un objet de type Class qui représente la classe à laquelle appartient l'objet appelant.
  - Pour afficher le nom de cette classe, il suffit d'appeler la méthode getName() sur l'objet retourné.

```
1    String S=getClasse().getName();
```

  - Pour créer un nouvel objet de cette même classe, on utilise la méthode newInstance().

```
1    Class c=getclasse();
2    try{ c.newInstance();
3    } catch (Exception e){ .... }
```

Les méthodes de la classe Object

- String toString() : retourne une représentation sous forme String de l'objet, affichable par l'intermédiaire de la méthode system.out.print(ln).
- La méthode toString() est appelée à chaque fois que l'on essai d'afficher un objet par la méthode system.out.print(ln).
- Il suffit donc de la redéfinir dans la classe de l'objet avec le comportement adéquat.

Exemple

```
1 class Point{
2 int x, y;
3 ...
4 public String toString(){
5 return("("+x+ ", "+y+ ") ");}}
```

Les classes internes

- Les classes internes (inner classes) sont une extension du langage Java introduite dans la version 1.1 du JDK.
  - Ce sont des classes qui sont définies dans une autre classe.
  - Les difficultés dans leur utilisation concernent leur visibilité et leur accès aux membres de la classe dans laquelle elles sont définies.

```
1 public class ClassePrincipale {
2 public class ClasseInterne { ... }
3 }
```

- Il est possible d'imbriquer plusieurs classes internes en Java.

```
1 public class ClassePrincipale {
2 public class ClasseInterne1 {
3 public class ClasseInterne2 {
4 public class ClasseInterne3 { ... }
5 } } }
```

Types de classes internes

- Il existe quatre types de classes internes :
  - les classes internes non statiques : elles sont membres à part entière de la classe qui les englobe et peuvent accéder à tous les membres de cette dernière.
  - les classes internes locales : elles sont définies dans un bloc de code. Elles peuvent être static ou non.
  - les classes internes anonymes : elles sont définies et instanciées à la volée sans posséder de nom.
  - les classe internes statiques : elles sont membres à part entière de la classe qui les englobe et peuvent accéder uniquement aux membres statiques de cette dernière.

Héritage d'une classe interne

- Une classe peut hériter d'une classe interne. Dans ce cas,
  - il faut obligatoirement fournir aux constructeurs de la classe une référence sur la classe principale de la classe mère et
  - appeler explicitement dans le constructeur le constructeur de cette classe principale avec une notation particulière du mot clé super.

```
1 public class ClassePrincipale{
2 public class ClasseInterne{ }
3 public class ClasseFille extends
  ClassePrincipale.ClasseInterne{
4 public ClasseFille(ClassePrincipale cp) {cp. super();}}}
```

- Une classe interne peut être déclarée avec final et abstract.
  - Avec le modificateur final, la classe interne ne pourra être utilisée comme classe mère.
  - Avec le modificateur abstract, la classe interne devra être étendue pour pouvoir être instanciée.

Classes internes anonymes

- Les classes internes anonymes (anonymous inner-classes) sont des classes internes qui ne possèdent pas de nom.
  - Elles ne peuvent donc être instanciées qu'à l'endroit où elles sont définies.
- Une syntaxe particulière de l'opérateur new permet de déclarer et instancier une classe interne :

```
1 new classe_ou_interface () {
2 // définition des attributs et des méthodes de la classe
  interne
3 }
```

- Une classe interne anonyme peut soit hériter d'une classe soit implémenter une interface mais elle ne peut pas explicitement faire les deux.

Classe interne anonyme

- Une classe interne anonyme ne peut pas avoir de constructeur puisqu'elle ne possède pas de nom mais elle peut avoir des initialisateurs.

```
1 public void init(){
2  boutonQuitter.addActionListener(
3  new ActionListener() {
4  public void actionPerformed(ActionEvent e) {
5  System.exit(0);
6 }});
7 }
```

- La classe interne anonyme ne pourra être instanciée dans le code qu'à l'endroit où elle est définie : elle est déclarée et instanciée en un seul et unique endroit.

Classes internes statiques

- Les classes internes statiques (static member inner-classes) sont des classes internes qui ne possèdent pas de référence vers leur classe principale.
  - Elles ne peuvent donc pas accéder aux membres d'instance de leur classe englobante.
  - Elles peuvent toutefois avoir accès aux variables statiques de la classe englobante.
  - Pour les déclarer, il suffit d'utiliser en plus le modificateur static dans la déclaration de la classe interne.

```
1 public class ClassePrincipale{
2public static class ClasseInterne {
3 public void afficher() {
4 System.out.println("bonjour");}}
5public static void main(String[] args) { 6
new ClasseInterne().afficher();}}
```

Exceptions en Java

- Une exception est une erreur qui se produit pendant l'exécution d'un programme.
- Ce sont des instances de classes dérivant de

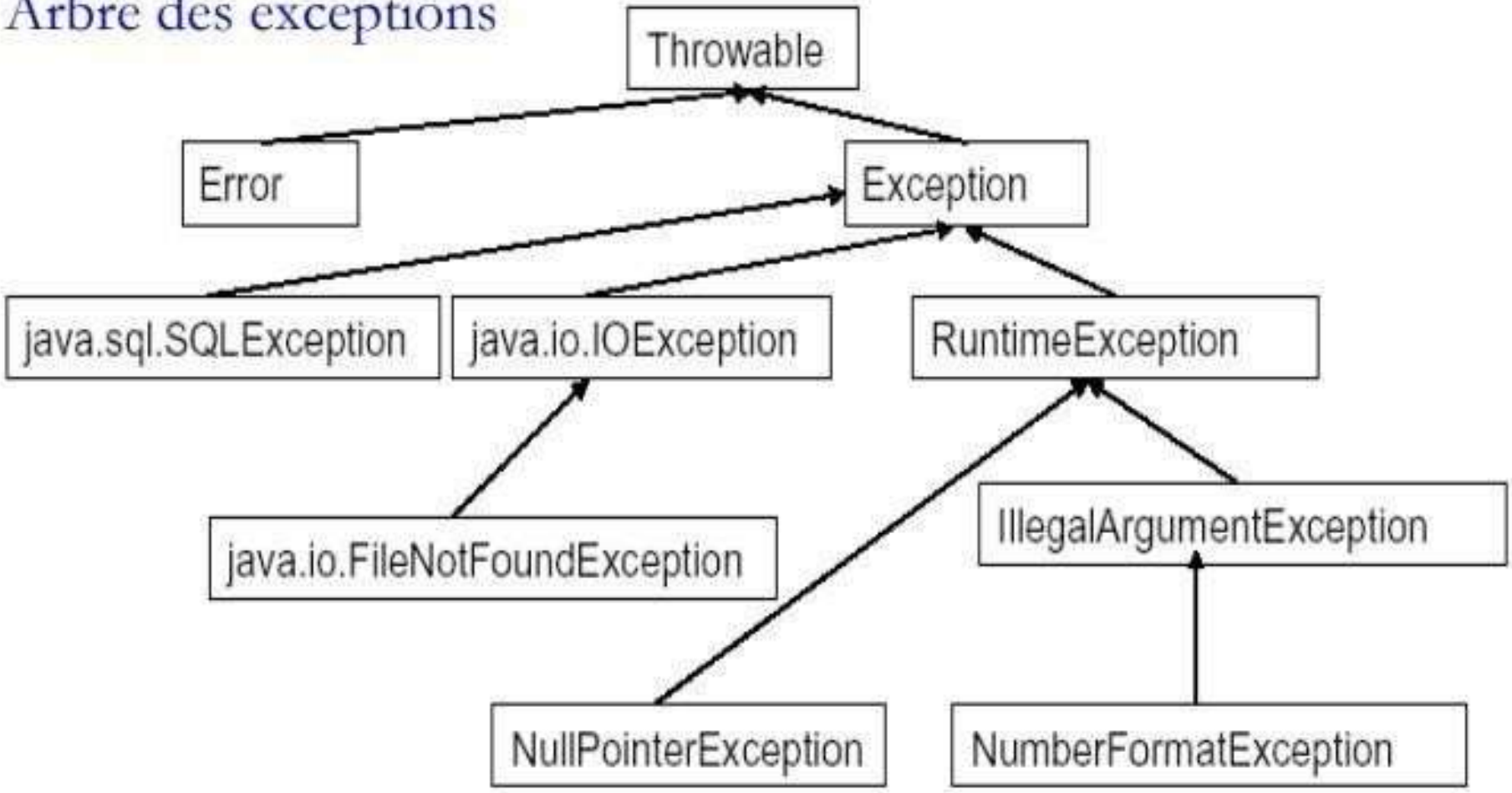
```
1    java.lang.Exception
```

- Elle peut conduire soit à l'arrêt du programme soit à des résultats incorrects.
- Les exceptions peuvent être générées
  - soit par l'environnement d'exécution Java, par exemple manque de place mémoire,
  - soit à cause du non respect des règles du langage ou
  - soit par le code (exceptions définies explicitement par les utilisateurs).

Gestion des exceptions

→ Toute exception en Java est un objet instancier d'une sous classe de la classe Exception

→ Arbre des exceptions



Attraper des exceptions système

- Il est possible d'attraper (capturer) l'exception, à l'aide du bloc « try-catch», afin de gérer cette exception.

```
1try { // bloc du code décrivant le déroulement normal du
     programme et les instructions de ce bloc sont
     susceptibles de lever une exception
2}
3 catch (ExceptionType1 objetTypeException) {
4 // Ce bloc constitue le gestionnaire de l'exception de
  type ExceptionType1 et contient les instructions à
  exécuter en cas d'erreur
5}
6 catch (ExceptionType2 objetTypeException) {
7 // Ce bloc constitue le gestionnaire de l'exception de
  type ExceptionType2 et contient les instructions à
  exécuter en cas d'erreur
8}
```

Exemple 1 (division par zéro)

Main.java

```
1  public class TestDivZero {
2  public static void main(String[] args) {
3  try{
4  int a = 2;
5  int b=2;
6  6 int c = 100 / (a b); }
7  catch( ArithmeticException e) {
8  System.out.println("Attention : division par Zéro"); }
9  9 }
10 10 }
11
```

```
Attention : division par Zéro
```

Bloc finally

- finally c'est une instruction optionnelle qui est exécutée quelle que soit le résultat du bloc try (c'est-à-dire qu'il ait déclenché ou non une exception).
- Elle permet de spécifier du code dont l'exécution est garantie quoi qu'il arrive.

```
1 try {
2 . . .
3 }
4 catch (...) {
5 . . .
6 }
7 finally {
8 . . .
9 }
```

Constructeurs des exceptions

- La création d'exception personnalisée peut être réalisée en rajoutant des constructeurs et des membres supplémentaires.
- Par convention, toutes les exceptions doivent avoir au moins deux constructeurs :
  - un sans paramètre.
  - un autre dont le paramètre est une chaîne de caractères utilisée pour décrire le problème.

```
1 public class MonException extends Exception {
2 public MonException(String text) { super(text); }}
```

- On utilise le mot clé throw pour lever une exception en instanciant un objet sur une classe dérivée de la classe Exception.

```
1 throw new MonException("blabla");
```

Constructeurs et méthodes de la classe Throwable

- Exception() : constructeur sans argument.
- Exception(String) : Constructeur avec Argument.
- getMessage() : retourne le message d'erreur décrivant l'exception.
- printStackTrace() : affiche sur la sortie standard la liste des appels de méthodes ayant conduit à l'exception.

Les chaînes de caractères

- Une chaîne de caractères (appelée String) est une suite (une séquence) de caractères.
- Longueur et accès aux caractères d'une chaîne de type String

```
String s1="bonjour";
String s2=""; // chaîne vide
int n1=s1.length( ); // n1 = 7
int n2=s2.length( ); // n2 = 0
```

- La méthode charAt( ) de la classe String permet d'accéder à un caractère de rang donné d'une chaîne.

```
1 String s ="bonjour";
2 char c1= s.charAt(0); // c1='b'
3 char c2= s.charAt(2); // c2='n'
```

Déclaration et création

- String s; : déclare que s est une référence à un objet de type String. Ici on a créé la référence mais pas l'objet.
- String n'est pas un type primitif, pour créer l'objet, on utilise l'opérateur new.

```
1 String s;
2 s = new String ("bonjour"); //s contient la référence à un objet contenant la chaine "bonjour".
```

- On peut regrouper les deux instructions :

```
1 String s=new String ("bonjour");
```

- On peut aussi écrire tous simplement :

```
1 String s= "bonjour";
```

Caractéristiques

- Un objet de type String n'est pas modifiable :
  - Il n'existera aucune méthode permettant de modifier le contenu d'un objet de type String.
  - Attention, l'affectation ne modifie l'objet.

```
1    String s;
2    s="bonjour";
3    s="bonsoir";
```

  - L'objet sn'a pasété modifié, mais c'est la référencesqui aété modifiée.
  - Après affectation, s référence un autre objet (la chaîne "bonsoir")

Concaténation de chaînes

- La méthode concat() permet la concaténation de deux chînaes de caractères.

```
1 String ch1= "le langage";
2 String ch2= "java";
3  String ch=ch1.concat( ch2); // ch contient "le langage java"
```

- Pour la concaténation, on peut aussi utiliser l'opérateur + de la manière suivante : Soient s1 et s2 deux objets de type String.

```
1 String s =s1+s2;
2 // équivalent à    String s=s1.concat(s2);
```

D'autre opération sur les chaines

- La méthode indexOf() retourne la position de la première occurrence d'un caractère dans une chaîne de caractères.
- La recherche commence à partir du début de la chaînes.Si on ne trouve pas de caractères, dans la chanes, correspondant au caractère ch, elle retourne 1.
- Comparaisons des références
- s1.equals(s2); retourne true.
  s1.equals(s3); retourne true. s1 et s2 ont le même contenu mais ils référencent deux objets différents.
  Remplacement de caractères : La méthode replace() crée une nouvelle chaîne en remplaçant toutes les occurrences d'un caractère donné par un autre.
- Extraction de sous chaîne:La méthode substring( ) permet d'extraire de sous chaîne d'une chaîne de caractères.
- String substring (int début) ; retourne un nouveau String qui commence au caractère d'indice début et allant jusqu'au dernier caractère de la chaîne s.
- Conversion en majuscule ou en minuscule: La méthode toLowerCase( ) crée une nouvelle chaîne en remplaçant toutes les majuscules par leur équivalent en minuscules. La méthode toUpperCase( ) crée une nouvelle chaîne en remplaçant toutes les minuscules par leur équivalent en majuscules.

Introduction

- En POO, la généricité est un concept permettant de définir des algorithmes (types de données et méthodes) identiques qui peuvent être utilisés sur de multiples types de données.

- La généricité permet de paramétrer du code avec des typesde données.
  Exemple:Class ArrayList<T>dont le code est paramétré par un type T

Classegénérique

Une classe générique est une classe comprenant une ou plusieurs variables de type = une classe paramétrée, dont le paramètre est un type d'objet

Exemple:

```
public class Paire<T>{
private T premier;
private T deuxième;
public Paire() { premier=null; deuxième=null;}
public Paire(T premier, T deuxième) {
this.premier=premier;
this.deuxième=deuxième;}
   public T getPremier() {return this.premier;}
public void setPremier(T premier) { this.premier = premier;}
public T getDeuxième() {return this.deuxième;}
public void setDeuxième(T deuxième) {this.deuxième = deuxième;
}
```

Classegénérique

La classe Paire introduit une variable de type T, incluse entre <> après le nom de la classe
Une classe générique (paramétrée) peut posséder plusieurs variables de type<T1,T2,T3,….>
Les variables de type peuvent être utilisées tout au long de la définition de la classe pour spécifier le type de retour des méthodes, le type des attributs, ou même le type de certaines variables locales rien n'empêche également d'utiliser des attributs ou d'autres éléments avec des types bien définis, c'est-à-dire non paramétrable, comme int, String, etc

- Utilisation de la classe générique:

-L'attribut premier utilise une variable de type
-Ainsi, commeson nom l'indique premier pourra être de n'importe quel type
-C'est celui qui utilisera cette classe qui spécifiera le type qu'il désire utiliser pour décrire l'objet de cette classe Paire

Classegénérique

Utilisation de la classe générique:

- Par exemple :

```
Paire<String> ordre;
```

Ici, ordre est un objet de type Paire<String>Le paramètre de la classe Paire, ou exprimé autrement, la variable de type est String Ainsi, au moment de cette déclaration, à l'intérieur de la classe Paire, tout ce qui fait référence à T est remplacé par le véritable type, c'est-à-dire String.

```
public class Paire {private String premier ; private String deuxième;
public Paire() { premier=null; deuxième=null;}
public Paire(String premier, String deuxième) {
this.premier=premier; this.deuxième=deuxième;}
public String getPremier() {return this.premier;}
public void setPremier(String premier) { this.premier = premier;}
 public String getDeuxième() {return this.deuxième;}
public void setDeuxième(String deuxième) {this.deuxième =
deuxième; }}
```

Classegénérique

Le code qui utilise la généricité a de nombreux avantages par rapport au code non générique: Contrôles de type plus forts au moment de la compilation. Un compilateur Java applique un contrôle de type fort au code générique et émet des erreurs si le code enfreint la sécurité au niveau de typage. Il est plus facile de corriger les erreurs de compilation au lieu de corriger les erreurs d'exécution, ce qui peut être difficile à trouver.

Variabledetype

- Par convention, les noms de paramètre de type générique sont des lettres simples majuscules. Pour une bonne raison: sans cette convention, il serait difficile de faire la différence entre une variable de type et un nom de classe ou d'interface.
  Les noms de paramètre de type générique les plus couramment utilisés sont:
  T:est utilisé pour qu'une classe générique renvoie et accepte n'importe quel type d'objet
  N:est utilisé pour qu'une classe générique renvoie et accepte un nombre

.

Méthodegénérique

Les méthodes sur java vont nous permettre de créer des algorithmes de code que nous allons pouvoir faire appel quand on en a besoin.

Les méthodes génériques sont celles qui sont écrites avec une seule déclaration et qui peuvent être appelées avec des arguments de différents types.

En fonction des types d'arguments passés à la méthode générique, le compilateur gère chaque appel de méthode de manière appropriée.

Méthodegénérique

Syntaxe d'une méthode générique

- Les méthodes génériques ont un type paramétré(<T>)avant le type de retour de la déclaration de méthode.
- Les méthodes génériques peuvent avoir différents types paramétrés séparés par des virgules dans la signature de méthode : <T1,T2,…>
- Le corps d'une méthode générique est comme une méthode normale

---

Appel de la méthode générique

On peut appeler une méthode paramétrée en la préfixant par le (ou les) type qui doit remplacer le paramètre de type .(<T>)

Méthode générique

```
1 package generique;
2
3 public class Test {
4     public <T> void affiche(T val) {System.out.println("valeur="+val);}
5
6     public <E,T> void afficher(E p1,T p2) {System.out.println(p1+" - "+p2);}
7
8     public static void main(String[] args) {
9
10        Test obj=new Test();
11        obj.affiche("bonjour");obj.affiche(1);obj.affiche(1.1);
12        System.out.println("-----------------------------------------------");
13        obj.afficher(1,"bonjour");obj.afficher(1.1,"bonjour");obj.afficher(1, 1.2);
14    }
15 }
16
```

Interfacegénérique

Les interfaces génériques ont les mêmes objectifs que lesinterfaces régulières. Ils sont soit créés pour exposer les membres d'une classe qui seront utilisés par d'autres classes, soit pour forcer une classe à implémenter des fonctionnalités spécifique.

=>Alors la généricité fonctionne également avec les interfaces.

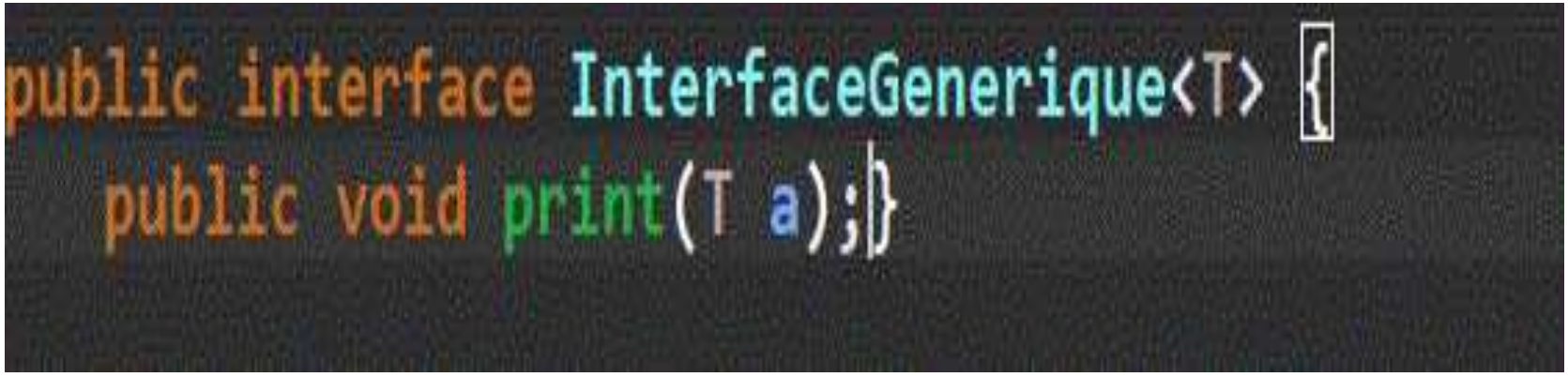
Syntaxe d'une interface générique

- Les interfaces génériques ont un type paramétré(<T>) après le nom de l'interface.
- Les interfaces génériques peuvent avoir différents types paramétrés séparés par des virgules : <T1,T2,…>
- Le corps d'une interface générique est comme une interface normale (Toutes les méthodes d'une interface sont implicitement abstraites).

Interfacegénérique

Déclaration des interfaces générique

Les interfaces génériques sont déclarées comme des classes génériques. Par exemple :



Implémentation des interfaces génériques

```
public class GenericClass<E> implements GenericInterface<E>
```

RQ: Une classe générique peut également avoir d'autres paramètres en dehors du paramètre de type qu'elle doit utiliser en raison de l'implémentation d'une interface générique.

```
public class GenericClass<K, V, E> implements GenericInterface<E>
```

Interfacegénérique

```java
package generique;
public interface InterfPaire<E,T> {
    public E getKey();
    public T getValue();

}
```

```java
package generique;


public class Paire<E,T> implements InterfPaire<E,T> {
    public E key;
    public T value;
    public Paire(E key,T value){this.key=key; this.value=value;}//constructeur
    public E getKey() {return key;}
    public T getValue() {return value;}
}

```

Interfacegénérique

```
1  package generique;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  public class Main {
7
8      public static void main(String[] args) {
9          InterfPaire<String,Integer> paire1=new Paire<String,Integer>("age",20);
10         System.out.println(paire1.getKey()+" = "+paire1.getValue());
11         InterfPaire<String,String> paire2=new Paire<String,String>("user","root");
12         System.out.println(paire2.getKey()+" = "+paire2.getValue());
13
14         List<String> LIVRE=new ArrayList<>();
15         LIVRE.add("Les misérables");
16         LIVRE.add("Victor HUGO");
17         InterfPaire<String,List<String>> paire3=new Paire<String,List<String>>("livre",LIVRE
18         System.out.println(paire3.getKey()+" = "+paire3.getValue());
19
20
21     }
22
23 }
```

```
Console ×  Coverage
<terminated> Main (4) Java Application C:\Users\...\eclipse\plugins\org
age = 20
user = root
livre = [Les misérables, Victor HUGO]
```

Diamant

Dans java SE 7 et ultérieur l'argument type peut être remplacé par un ensemble vide d'arguments de type (<>) appelé le diamant (Diamond)
Version >= java SE 7
Pour instancier une classe générique on écrit :
Avec le **diamant :** Paire<Integer> P1= new Paire<> () ;
Avec les anciens versions :Paire<Integer> P1= new Paire<Integer> ();
La seule limite du diamant est si on l'utilise avec les classes anonymes

```
1 package pack2;
2
3 public  abstract class ClasseAnonyme<T> {
4     public abstract boolean compare(T arg1,T arg2);
5
6 }
7
```

```
1
2 public class TestDiament {
3
4     public static void main(String[] args) {
5         // TODO Auto-generated method stub
6         ClasseAnonyme<String> C1=new ClasseAnonyme<String>()
7         {};
8
9     }
10
11 }
12
```

Diamant

```
1
2  public class TestDiament {
3
4      public static void main(String[] args) {
5          // TODO Auto-generated method stub
6          classeAnonyme<String> C1=new classeAnonyme<String>()
7          {
8
9              @Override
10             public boolean compare(String arg1, String arg2) {
11                 // TODO Auto-generated method stub
12                 return arg1.equals(arg2);
13             }};
14             System.out.println(C1.compare("Bonjour", "Bonsoir"));
15
16     }
17
18 }
19
```

ExtendsA&B

On sait bien que l'Héritage multiple n'est pas autorisé dans java mais il y a une solution qui collectionne entre la généricité ,l'héritage et les interfaces.

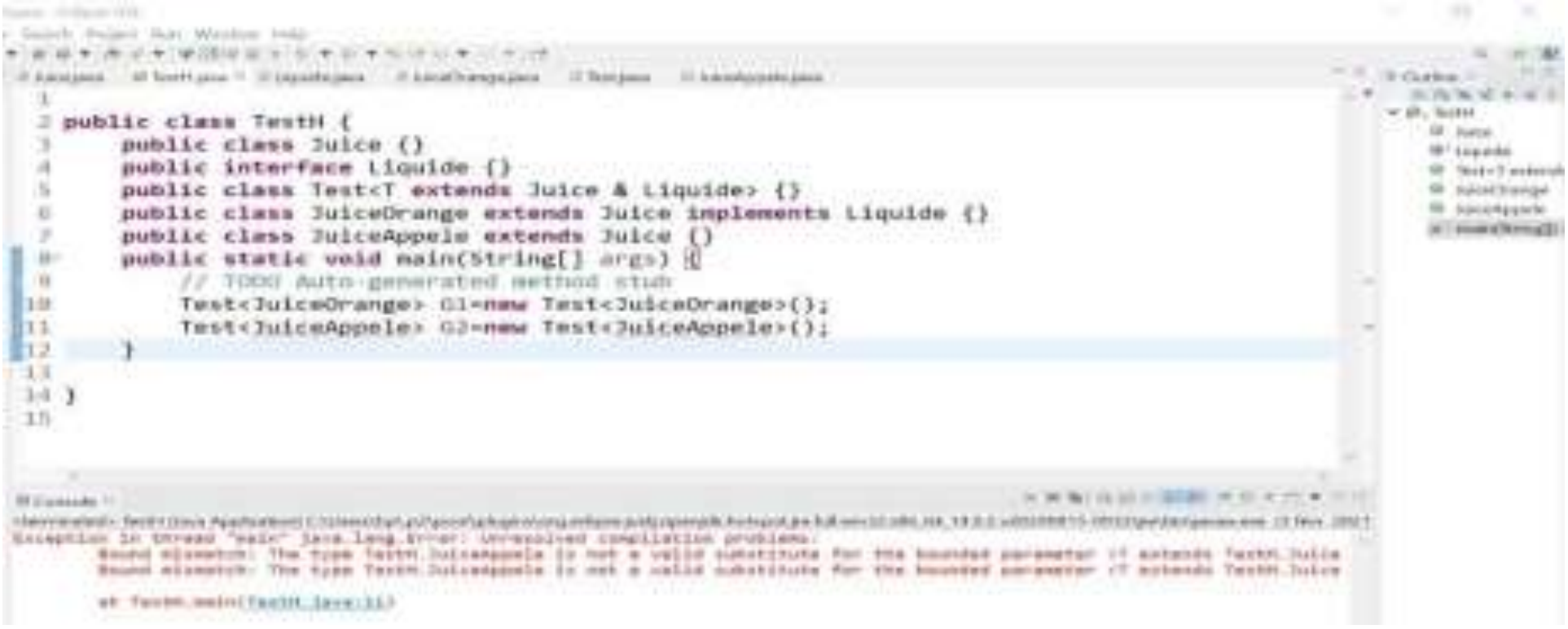
```
public interface Liquide{…}
public class Jus{…}
public class TestJus<T extends Jus & Liquide>{}
```

On a le droit d'instancier la classe TestJus à condition que T doit être une classe qui hérite de Jus et implémente Liquide

```
public class JusOrange extends Jus implements Liquide{}

TestJus<JusOrange> G1=new TestJus< JusOrange> ();
```

Extends A & B



```
public class TestH {
    public class Juice {}
    public interface Liquide {}
    public class Test<T extends Juice & Liquide> {}
    public class JuiceOrange extends Juice implements Liquide {}
    public class JuiceAppele extends Juice {}
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Test<JuiceOrange> G1=new Test<JuiceOrange>();
        Test<JuiceAppele> G2=new Test<JuiceAppele>();
    }

}
```

Définition

Une collection est un objet dont la principale fonctionnalité est de contenir d'autres objets, comme un tableau.
Le JDK fournit d'autres types de collections sous la forme de classes et d'interfaces.
Ces classes et interfaces sont dans le paquetage java.util.

Plusieurs types de collections proposés:
List : Collection ordonnée, aussi appelée séquence.
Queue : Collection pour la gestion d'éléments avant leur traitement.
Exemple : file d'attente LIFO, FIFO ou avec un tri défini.
Set : ensemble d'élément non ordonné, un élément peut être présent une ou plusieurs fois selon les mises en œuvre.
SortedSet : Ensemble d'éléments triés selon un tri défini.

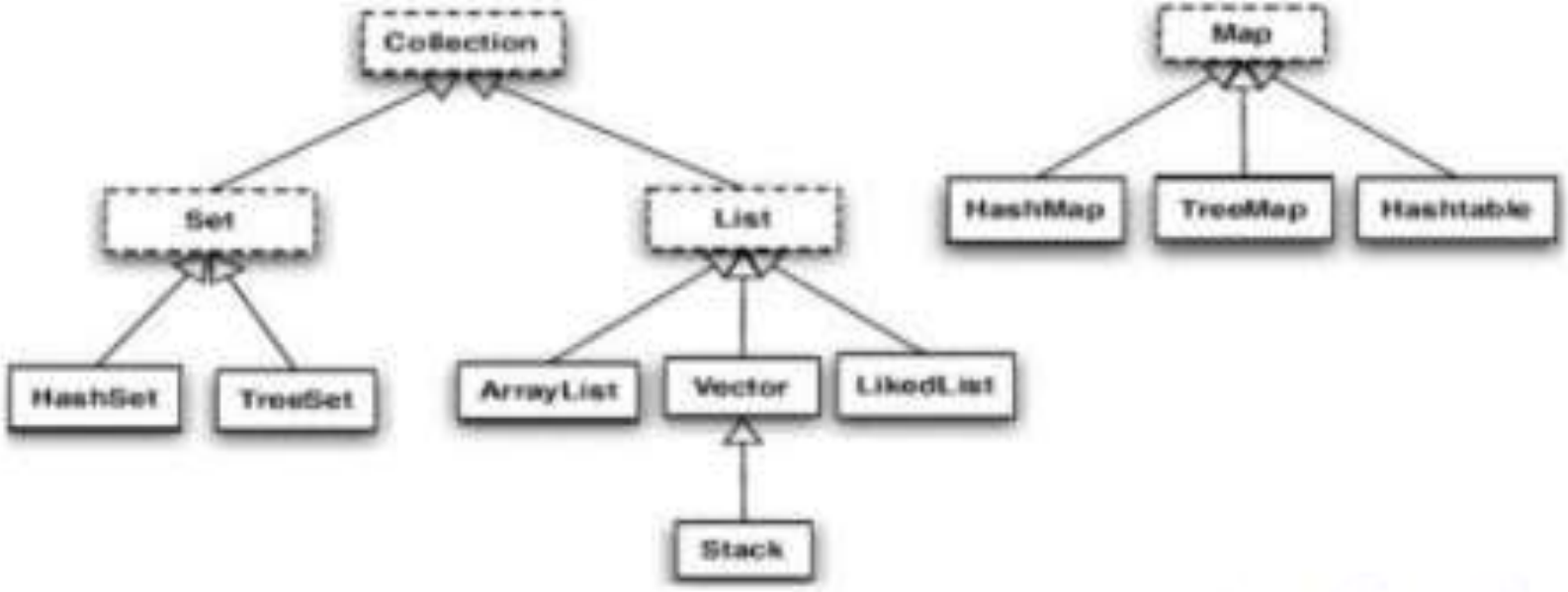
Tous imports de ce chapitre sont de java.util.*;

Hiérarchie des interfaces

Des interfaces dans deux hiérarchies d'héritage principales :

- Collection<E> : correspond aux interfaces des collections proprement dites.
- Map<K,V> : correspond aux collections indexées par des clés; un élément de type V d'une map est retrouvé rapidement si on connaît sa clé de type K (comme les entrées de l'index d'un livre).

Les collections Java contiennent des éléments de type Object.



List:ArrayList

- Pas de limite de taille
- Possibilité de stocker tout type de données (y compris null)
- Pour créer un ArrayList :

```
ArrayList list = new ArrayList();
```

- Pour ajouter la valeur 3 a la liste :

```
list.add(3);
```

- Pour récupérer la taille de la liste:

```
System.out.println(list.size());
```

- Pour récupérer un élément de la liste

```
System.out.println(list.get(indice))
```

List:LinkedList

C'est une liste dont chaque élément a deux référerences : une vers l'élément précédent et la deuxième vers l'élément suivant.

- Pour le premier élément, l'élément précédent vaut null
- Pour le dernier élément, l'élément suivant vaut null

addFirst(object) : insère l'élément object au début de la liste

addLast(object) : insère l'élément Object comme dernier élément de la liste (exactement comme add())

Map:Hashtable

Un HashTable est une implémentation de Map qui associe une clé à une valeur. N'importe quel objet, mis à part null peut y être ajouté. Voici un exemple d'utilisation de HashTable, où on associe un numéro de mois à son nom :

```
Map monHashtable = new Hashtable() ;
monHashtable.put(new Integer(1),"Janvier");
monHashtable.put(new Integer(2),"Fevrier");
monHashtable.put(new Integer(3),"Mars");
monHashtable.put(new Integer(4),"Avril");
monHashtable.put(new Integer(5),"Mai");
monHashtable.put(new Integer(6),"Juin");
```

Map:HashMap

Un HashMap est une collection similaire au HashTable. Seules deux choses les diffèrent : HashMap accepte comme clé et comme valeur null

```
HashMap<String, Integer> map = new HashMap<>();

    // Ajouter les éléments

    map.put(« ALi", 2);//Combien de Ali on a dans la classe
    map.put(« Ahmed",3);
    map.put(« Fatima",5);
```

Set:HashSet

HashSet est l'implémentation la plus utile de Set. Elle permet de stocker des objets sans doublons, des exceptions sont levées si l'on essaie d'insérer un objet non autorisé (comme null). Cependant, si l'on essaie d'ajouter un objet déjà présent, la fonction add renverra tout simplement *false* et ne fera rien. Pour parcourir un HashSet

```
Set monHashSet=new HashSet(); // on crée notre Set
monHashSet.add(new String("1")); // on ajoute des string quelconques
monHashSet.add(new String("2"));
 monHashSet.add(new String("3"));
```

Présentation des processus & Threads

- Processus : est un programme qui s'exécute et qui possède son propreespacemémoire: ses registres, ses piles, ses variables et sonpropreprocesseurvirtuel

- Thread : Les threads Java sont des processus légers : ils partagentun ensemble decodes, donnéeset ressourcesau sein d'un processuslourd qui les héberge(ce processusest laJVM).

- Avantages des processus légers par rapport aux processus système:
  - rapidité de lancement et d'exécution;
  - partage des ressources système du processus englobant;
  - simplicité d'utilisation.

Création des threads

- Ils existe deux méthodes pour créer un Thread en Java :
  - La première méthode consiste à étendre la classe Thread du package java.lang et à redéfinir sa méthode run().
  - La deuxième méthode consiste à implémenter l'interface Runable par l'intermédiaire de sa méthode run() et de créer un Thread à base de la nouvelle classe.
- La classe java.lang.Thread et l'interface java.lang.Runnable sont les bases pour le développement des threads en java.
- Par exemple, pour exécuter des applets dans un thread, il faut que celles ci implémentent l'interface Runnable.

Création d'un thread par extension de la classe Thread

- La classe Thread offre la possibilité de créer un processus qui se termine immédiatement après sa création.
  - car la méthode run() exécutée par le processus dès qu'il est créé est définie vide dans la classe Thread.
- Pour personnaliser le comportement des processus à créer, il suffit donc d'étendre la classe Thread avec redéfinition de la méthode run().
- Ensuite, la création d'un processus revient à instancier un objet Thread et à le faire démarrer comme processus à l'aide de la méthode start() définie dans la classe Thread.
  - Celle-ci crée le processus et appelle automatiquement la méthode run().

Exemple

```
1  import java.lang.Thread.*;
2  class Processus extends Thread {
3  public void run() {
4  // Comportement du procssus
5  }
6  }
7  public class CREATION {
8  public static void main(String[] args) {
9  Processus p =new Processus();
10 p.start();
11 }
12 }
```

Remarques

- La création d'un objet Processus n'implique pas la création du thread. Le processus doit être créé explicitement par un appel à la méthode start() sur l'objet créé ; celle-ci réalise deux tâches principales :
  - La première tâche consiste à faire démarrer le processus en tant que processus.
  - La deuxième tâche consiste à appeler la méthode run().
- Appeler la méthode run() à la place de start() ne fait pas démarrer le processus, mais exécute tout simplement la procédure run() de la classe Processus.
- La classe Thread dispose de 8 constructeurs dont l'un est un constructeur à un paramètre de type String permettant d'affecter un nom au processus créé :

```
1    public Thread (String name)
```

Création d'un thread par implémentation de l'interface Runnable

- Un thread peut être créé par implémentation de l'interface Runnable du package java.lang. Ceci est réalisé par l'intermédiaire de l'un des deux constructeurs suivants qui reçoivent en paramètre un objet d'une classe qui implémente l'interface Runnable :

```
1 public Thread(Runnable target)
2 public Thread(Runnable target, String name)
```

- L'interface Runnable déclare une seule méthode « run() » que la classe de l'objet 'target' doit implémenter.
  - La méthode run() sera alors exécutée par le thread créé.
- Noter que pour faire référence au Thread en cours d'exécution au sein des méthodes de votre classe implémentant Runnable ; on utilise la méthode statique Thread.currentThread().

Création d'un processus

- La création d'un processuspassepar deux étapes:
  - La première étape consiste à implémenter l'interface Runnable :

```
1        class Traitement implements Runnable{
2        Public void run(){ // comportement du
             processus
3        }
4        }
```

  - La 2ème étape consiste à créer un thread à base d'un objet de la classe précédente:

```
1        Traitement tr=new Traitement();
2        Tread P=new Thread ( t r , "Nom");
3        P.start();
```

Moniteur et ressource

- Une ressource critique est une ressource qui ne doit être accédée que par un seul thread à la fois.
- En Java, il n'est pas possible de contrôler directement l'accès simultané à une variable, il faut l'encapsuler et contrôler l'exécution de l'accesseur correspondant.
- Un moniteur est un verrou qui ne laisse qu'un seul thread à la fois accéder à la ressource.
- En Java, tout objet peut jouer le rôle de moniteur. On pose un moniteur sur un bloc de code à l'aide du mot clé synchronized.

```
1 synchronized(objetMonitor){
2 ... //code en section critique
3 }
```

Exemple

```
1  class MoniteurImpression {
2  synchronized public void imprime(String t) {
3  for (int i=0; i<t.length(); i++) {
4  System.out.print(t.charAt(i));
5  try { Thread.currentThread().sleep(100); }
6  catch (InterruptedException e) {};
7  }}}
8  class TPrint extends Thread {
9  String txt;
10 static MoniteurImpression mImp = new MoniteurImpression();
11 public TPrint(String t) {
12 txt = t;
13 }
14
```

Exemple (suite)

```
15  public void run() {
16  for (int j=0; j<3; j++)
17  mImp.imprime(txt);
18  } }
19  public class TestThread4 {
20  static public void main(String args[]) {
21  TPrint a = new TPrint("bonjour ");
22  TPrint b = new TPrint("au revoir ");
23  a.start();
24  b.start();
25  }}
26
27
```

```
bonjour bonjour bonjour au revoir au revoir au revoir
```

Synchronisation des méthodes

- Les méthodes bénéficier de la synchronisation, afin d'éviter que plusieurs threads n'y accèdent en même temps.
- Si une classe Class1 disposed'une méthode par exemple: void sc() qui doit être exécuté en exclusion mutuelle.
  - Celle-ci doit alors être insérée dans un moniteur par l'intermédiaire du modificateur synchronized de la manière suivante :

```
1    Class Class1{
2    ...
3    synchronized void sc(){...}
4    ... }
```

- Si un processus est alors en train d'exécuter la méthode sc() sur un objet m (m.sc()), aucun autre processus n'aura le droit d'exécuter la même procédure sur le même objet m jusqu'à ce que le premier processus quitte la méthode.