Langage Java
et systèmes distribués

UIC – 2ème GI

Karim GUENNOUN

Points abordés

- Héritage et Interfaces
- Threads
- Communication par sockets
- RMI

Héritage

Concept

- L'héritage représente la relation: EST-UN
- Exemples:
  - Un Technicien est un Employé…
- La classe dont on dérive est dite classe de base ou classe mère :
  - Employé est la classe de base (classe supérieure),
- les classes obtenues par dérivation sont dites classes dérivées ou classes filles :
  - Technicien, Ingenieur et Directeur sont des classes dérivées (sous-classes).

Exemple

- package Test2018;
- public class Employe
- {
- String Nom;
- String Prenom;
- double Salaire;
- public String Afficher()
- {
- String s="Nom: "+ nom+", Prenom: "+ prenom+ " salaire: " + salaire;
- return(s); } }

Intérêt 1/2

- La réutilisation:
  - Profiter d'une classe déjà codée pour en définir une nouvelle
  - Optimisation de code: Pas besoin de reprendre les attributs et méthodes définis dans la classe mère.
  - package Test2018;
  - public class ChefService extends Employe
  - {
  - String NomService;
  - double PrimeEncadrement;

Intérêt 2/2

- La factorisation de code:
  - Factoriser des attributs et des méthodes en communs entre certaines classes
  - Ecrire ces attributs/méthodes une seule fois
  - Exemple: ChefDivision / ChefService

Classe dérivée

- Une classe dérivée modélise un cas particulier de la classe de base, et est enrichie d'informations supplémentaires.
- La classe dérivée possède les propriétés suivantes:
  - contient les attributs de la classe de base,
  - peut posséder de nouveaux attributs,
  - possède les méthodes de sa classe de base
  - peut redéfinir (masquer) certaines méthodes,
  - peut posséder de nouvelles méthodes

Héritage en Java

- Syntaxe: public class B **extends** A
- Une classe ne peut hériter que d'une seule classe à la fois. Il n'existe pas d'héritage multiple.
- Toutes les classes héritent de la classe Object. (package lang)
  - clone, equals et toString, getClass
  - Tout objet en Java peut être utilisé comme un verrou : wait, notify,…
  - ...

Le constructeur

- La classe dérivée doit prendre en charge la construction de la classe de base.
- Pour construire un Directeur, il faut construire d'abord un Employé
- Le constructeur de la classe de base est donc appelé avant le constructeur de la classe dérivée.
- Si un constructeur de la classe dérivée appelle explicitement un constructeur de la classe de base, cet appel doit être obligatoirement la première instruction de constructeur. Il doit utiliser pour cela, le mot clé **super**.

Héritage des constructeurs

- B hérite de A:
  - La classe A ne possède aucun constructeur ou possède un constructeur sans paramètre. Ce constructeur est alors appelé implicitement par défaut dans tous les constructeurs de B.
  - La classe A ne possède pas de constructeur par défaut et il n'existe que des constructeurs avec des paramètres. Alors, les constructeurs de B doivent appeler explicitement un des constructeurs de A.
  - L'appel super correspond forcément à la première ligne de code.

Droit d’accès

- Rappel, l’unité de protection est la classe:
  - public: les membres sont accessibles à toutes les classes,
  - «rien»: les membres sont accessibles à toutes les classes du même paquetage,
  - private: les membres ne sont accessibles qu'aux membres de la classe.
- Si les membres de la classe de base sont :
  - public ou « rien » : les membres de la classe dérivée auront accès à ces membres (champs et méthodes),
  - private : les membres de la classe dérivée n'auront pas accès aux membres privés de la classe de base (utiliser protected dans ce cas)

Les méthodes

- Redéfinition
  - Substituer une méthode par une autre
  - Même nom, même signature, même type de retour
  - public class ChefService extends Employe
  - {
  - String NomService;
  - double primeEncadrement;
  - public String Afficher()
  - {
  - String s=super.Afficher()+", NomService: "+NomService;
  - return(s); } }

Les méthodes

- Surdéfinition
  - Cumuler plusieurs méthodes ayant le même nom
  - Même nom mais signature différente
  - Ne doit pas diminuer les droits d'accès
    - …
    - public void Afficher(String Date)
    - {
    - System.out.println ("Nom: "+ nom+", Prenom: "+ prenom+ " salaire: " + salaire + ()+", NomService: "+NomService);
    - }
    - }

Compatibilité entre objets classe mère et classe fille

- Un objet de la classe fille est forcément un objet de classe mère
  - ChefService CS=new ChefService("toto", "titi",10000, "comptabilité",5000);
  - Employe E=new Employe(("toto", "titi",10000);
  - E=CS;
    - Conversion implicite: le compilateur recopie les attributs communs en ignorant le reste
  - CS=E;
    - Erreur: impossibilité de compléter les champs manquant
    - Obligation de caster pour passer la compilation: d=(Directeur) e;

Polymorphisme

- Permet de manipuler les objets sans connaitre leur type:
  - Employe[ ] tab=new Employe[3];
  - tab[0]=new Employe(….);
  - tab[1]=new Employe(…);
  - tab[2]=new ChefService(…);
  - for(i=0;i<0;i++)
  - {
  - System.out.println(tab[i].Afficher());
  - }

Méthodes et classe « final »

- Une méthode "final" ne peuvent être redéfinie dans les classes filles
- Les classes qui sont déclarées "final" ne peuvent posséder de classes filles

Récapitulatif

- Une classe hérite d'une autre classe par le biais du mot clé :extends.
- Une classe ne peut hériter que d'une seule classe.
- Si aucun constructeur n'est défini dans une classe fille, la JVM en créera un et appellera automatiquement le constructeur de la classe mère.
- La classe fille hérite de toutes les propriétés et méthodes public et protected.
- Les méthodes et les propriétés private d'une classe mère ne sont pas accessibles dans la classe fille.
- On peut redéfinir une méthode héritée
- On peut utiliser le comportement d'une classe mère par le biais du mot clé super.
- Si une méthode d'une classe mère n'est pas redéfinie ou « polymorphée », à l'appel de cette méthode par le biais d'un objet enfant, c'est la méthode de la classe mère qui sera utilisée.
- Vous ne pouvez pas hériter d'une classe déclarée final.
- Une méthode déclarée final n'est pas redéfinissable.

Classes abstraites et interfaces

Une classe abstraite, c'est quoi?

- Correspondent à des superclasses
- Utiliser principalement pour la conception ascendante
  - E.g : personne vis-à-vis : etudiant, enseignant, adminstratif
- Doit être déclarée avec le mot clé abstract
  - abstract class personne
- Permet de définir des méthodes sans donner leur corps
  - abstract void afficher();

Fonctionnement

- Une classe abstraite n’est pas instanciable
- Une méthodes abstraite ne peut être déclarée que dans une classe abstraite
- Les classes non abstraites héritant d’une classe abstraite doivent définir le comportement des méthodes abstraites
- S’il y en a plusieurs, le comportement est polymorphe

Une interface, c’est …

- Une interface est une classe 100% abstraite
- Une interface définit un contrat que certaines classes pourront s'engager à respecter.
- Une interface n’implémente pas les comportements, les comportements sont définis dans les classes d’implémentation
- Contient
  - des définitions de constantes
  - des déclarations de méthodes (prototype).

Interfaces et héritage

- L'héritage multiple est autorisé pour les interfaces
- Exemple
  - **interface** A { **void** f(); }
  - **interface** B **extends** A { **void** f1(); }
  - **interface** C **extends** A { **void** f2(); }
  - **interface** D **extends** B, C { **void** f3(); }

Implantation d’une interface

- Si une classe déclare qu'elle implémente une interface, elle doit proposer une implémentation des méthodes listées dans l'interface.
- La classe peut proposer des méthodes qui ne sont pas listées dans l'interface.

Implantation et héritage multiple

- Lorsqu'une interface hérite de plusieurs autres interfaces, il peut apparaître que des déclarations de méthodes de même nom sont héritées.
  - Si les deux déclarations de méthodes héritées ont des en-têtes identiques, il n'y a pas de problème.
  - Si les deux déclarations ont un même nom de méthode mais des paramètres différents, en types ou en nombre, la classe implémentant l'interface devra implémenter les deux méthodes.
  - Le problème apparaît lorsque deux déclarations de méthodes ont même nom et mêmes paramètres mais un type de retour différent. Dans ce cas, le compilateur Java refuse de compiler le programme car il y a un conflit de nom.

Quelques éléments…

- On ne peut pas instancier une interface en Java
- Le mot clé "**implements**" est utilisé pour indiquer qu'une classe implemente une interface
- L'implémentation de toute méthode appartenant à une interface doit être déclarée public
- La classe implementant une interface doit implémenter toutes les méthodes. Sinon, elle doit être déclarée abstract
- Les Interfaces ne peuvent être déclarées private ou protected
- Toutes les méthodes d'une interface sont par défaut abstract et public

Les threads

Processus

- Un processus est une unité d'exécution faisant partie d'un programme.
- Fonctionne de façon autonome et parallèlement à d'autres processus.
- Sur une machine mono processeur,
  - chaque unité se voit attribuer des intervalles de temps au cours desquels elle a le droit d'utiliser le processeur pour accomplir ses traitements.

Processus

- Le système alloue de la mémoire pour chaque processus :
  - segment de code (instructions),
  - segment de données (allocations mémoire),
  - segment de pile (variables locales).
- le système associe à chaque processus
  - identificateur
  - priorités,
  - droits d'accès…

Etats des processus

- **Nouveau** (new): le processus est créé mais pas encore démarré
- **Exécutable** (runnable): la méthode de démarrage a été appelée
- **En exécution** (running) : le processus s'exécute sur un processeur du système
- **En attente** (waiting) : le processus est prêt à s'exécuter, mais n'a pas le processeur (occupé par un autre processus en exécution) ;
- **bloqué** (blocked): il manque une ressource (en plus du processeur) au processus pour qu'il puisse s'exécuter ou est arrêté à cause par exemple d'un timer.
- **Terminé** (terminated): il termine (fin normale de l'exécution ou levée d'une exception)

Les threads

- Appelé aussi processus léger
- Un thread est un processus à l'intérieur d'un autre processus
- Les ressources allouées à un processus vont être partagées entre les threads qui le composent
- Chaque thread possède son propre environnement d'exécution (valeurs des registres du processeur) ainsi qu'une pile (variables locales).

Les threads Java

- L'utilisation/comportement des threads changent d'un OS à un autre
- Java définit son propre concept de thread:
  - La JVM permet l'exécution concurente de plusieurs threads
  - Chaque thread possède une priorité. Les threads de plus haute priorité s'exécutent en premier
  - La JVM démarre avec l'exécution du thread correspondant à la méthode main

Codage

- Un thread est un objet Java qui
  - est une instance d'une classe qui hérite de la classe Thread,
  - ou qui implémente l'interface Runnable.

Par héritage

- Déclaration
  - class MonThread extends Thread

    {

    public void run()

    { // traitement parallèle . . . }

    }
- Création
  - MonThread p = new MonThread ();
  - p.start();

Par implémentation

- Déclaration
  - class MonRunnable implements Runnable

```
{
public void run()
{ // traitement . . . }
}
```

- Création
  - MonRunnable p = new MonRunnable ();

```
Thread q = new Thread(p);
 q.start();
```

La classe Thread

- Fait partie du package java.lang
- Un constructeur sans arguments
- D'autres constructeurs pouvant considérer les arguments
  - Un nom: le nom du Thread (par défaut, le préfixe Thread- suivi par un numéro incrémenté automatiquement)
  - Un objet qui implémente l'interface Runnable: contient les traitements
  - Un group: le groupe auquel sera rattaché le Thread

Quelques méthodes

void destroy()	met fin brutalement au thread
String getName()	Renvoie le nom du thread
long getId()	renvoie un numéro donnant un identifiant au thread
int getPriority()	renvoie la priorité du thread
boolean isAlive()	renvoie un booléen qui indique si le thread est actif ou non
boolean isInterrupted()	renvoie un booléen qui indique si le thread a été interrompu
void join()	attend la fin de l'exécution du thread
void join(long m)	attend au max m millisecondes la fin de l'exécution du thread
void sleep(long m)	mettre le thread en attente durant le temps exprimé en millisecondes fourni en paramètre.
void start()	démarrer le thread et exécuter la méthode run()
…	…

Exemple (arrêt)

```
public class T extends Thread
{
public void run()
{
int i;
for(i=0;i<5;i++)
{
System.out.println("je suis vivant");
try{
this.sleep(1);
}
catch(Exception e)
{}}}
public static void main(String args[]) throws Exception
{
T t=new T();
t.start();
while(t.isAlive())
{
System.out.println("le processus est vivant");
}}}
```

Partager les variables entre Threads

- Les threads d'un même processus partagent le même espace mémoire.
- Chaque instance de la classe thread possède ses propres variables (attributs).
- Pour partager une variable entre plusieurs threads, on a souvent recours à une variable de classe (par définition partagée par toutes les instances de cette classe)

Exemple

```
public class Partager extends Thread
{               private static String nomGlobal= "";
                private String nomLocal;
                Partager ( String s ) { nomLocal = s; }
                public void run()
                {
                nomGlobal = nomGlobal + nomLocal;
                System.out.println(nomGlobal);
                }
                public static void main(String args[])
                {
                Thread T1 = new Partager( "Toto" );
                Thread T2 = new Partager( "Titi" );
                T1.start(); T2.start();
                System.out.println( nomGlobal );
                System.out.println( nomGlobal );
                System.out.println( nomGlobal );
                System.out.println( nomGlobal ); } }
```

Synchronisation: motivation

- Exemple

```
public class Tableau
{
int tab[];
int indice;
public Tableau(int[] T)
{
tab=T;
indice=0;
}

public void lire()
{
System.out.println("lu: "+tab[indice]);
indice++;
}
}
```

```
public class Lecteur extends Thread
{
public Tableau t;
public Lecteur(Tableau Tab)
{
t=Tab;
}
public void run()
{
t.lire();t.lire();t.lire();t.lire();t.lire();
}

public static void main(String args[])
{
int tabEntiers[]=new int[10];
for(int i=0;i<10;i++)
{
tabEntiers[i]=i;
}
Tableau tab=new Tableau(tabEntiers);
Lecteur L1=new Lecteur(tab);
Lecteur L2=new Lecteur(tab);
L1.start();
L2.start();}}
```

Exclusion mutuelle et section critique

- Utilisation de sections synchronisées:

```
public class Tableau
{
…
public void lire()
{
synchronized(this)
      {
         System.out.println("lu: "+tab[indice]);
          indice++;
      }
}
```

```
public class Tableau
{
…
public synchronized void lire()
{
System.out.println("lu: "+tab[indice]);
indice++;
}
}
```

- Interdit l'exécution de la section critique sur le même objet de manière entrelacée