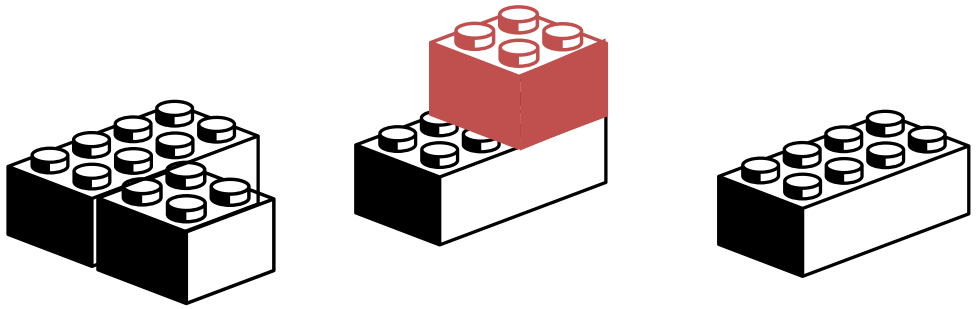
POO
Programmation Orientée Objets

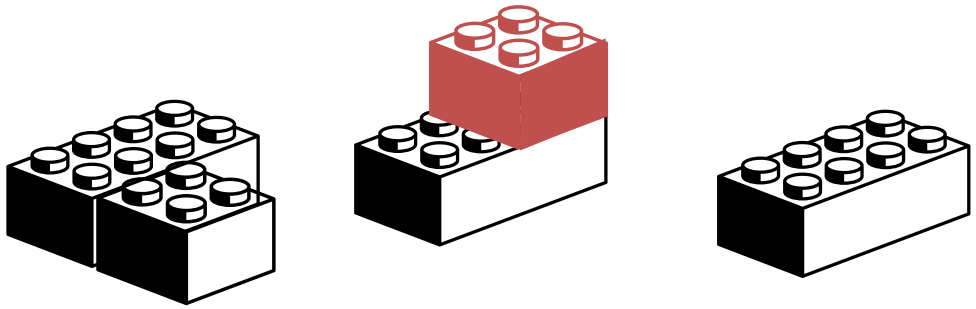
Mise en œuvre par le langage JAVA

J. Saraydaryan



Langage Java Avancé

J. Saraydaryan



Gestion des exceptions en JAVA

Gestion des exceptions

- Définition

Mécanisme permettant de gérer les erreurs en Java.

- Propriétés
  - Une exception sera représentée par un objet contenant l'erreur
  - Ensemble de mots clés pour détecter et traiter les erreurs
    - Try : début de la déclaration du bloc à surveiller
    - Catch : fin du bloc et traitement à effectuer en cas d'erreur
    - Finally : bloc de traitement s'effectuant après toutes les autres opérations

Gestion des exceptions

- Exemple d'exceptions

```
{
   int[] tab;
   tab= new int[3];
   int a,b,result;

   a=4654;
   b=0;

   result = a/b;
}
```

```
Exception in thread "main"
java.lang.ArithmeticException: / by zero
at
com.course.examples.exception.SampleExcept
ion.main(SampleException.java:10)
```

```
{
   int[] tab;
   tab= new int[3];
   int a,b,result;

   a=4654;
   b=0;

   result=tab[4];
}
```

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 4
at
com.course.examples.exception.SampleExcepti
on.main(SampleException.java:13)
```

Gestion des exceptions

- Détection d'exceptions

```
int[] tab;
tab= new int[3];
int a,b,result;

a=4654;
b=0;

try{
   result = a/b;
   result=tab[4];
}
catch (Exception e) {

   System.out.println("Exception:"
                      +e.getMessage());

}
```

Bloc de détection des erreurs

Type d'erreur détecté et géré, **e** contiendra les informations sur l'erreur détectée

Bloc de traitement lors de la détection d'une erreur

Gestion des exceptions

- Détection d'exceptions

```
try{
   // Zone de détection d'exception
}catch (<Type d'exception 1> e) {
   // Gestion de l'exception type 1
}catch (<Type d'exception 2> e) {
   // Gestion de l'exception type 2
}finally {
    // Bloc d'exécution finale
}
```

Gestion des exceptions

- Détection d'exceptions

```
int[] tab;
tab= new int[3];
int a,b,result;

a=4654;
b=0;

try{
   result = a/b;
   result=tab[4];
}catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("Erreur sur le tableau
                        e:"+e.getMessage());

}catch (ArithmeticException e) {
    System.out.println("Erreur sur les opération
                        e:"+e.getMessage());

}finally {
     System.out.println("Erreurs traitées,
                                fin du prog.");
}
```

Bloc de détection des erreurs

Déclenchement sur erreur de type **ArrayIndexOutOfBoundsException**

Déclenchement sur erreur de type **ArithmeticException**

Déclenchement à la fin de tous les traitements

Gestion des exceptions

- Délégation de la gestion de l'exception

Demande à la classe/méthode parente de gérer l'exception (**throw**)

```
public class Process {

public float divisionA(float a, float b){
  float result=0;
  try{
      result=a/b;
      return result;
   }catch (ArithmeticException e) {
      System.out.println("e:"+
                    e.getMessage());
      return 0;
   }
 }
}
```

Gestion Locale de l'exception

```
public class Process {

   public float divisionB(float a, float b)
                    throws ArithmeticException{
       return a/b;
  }
}
```

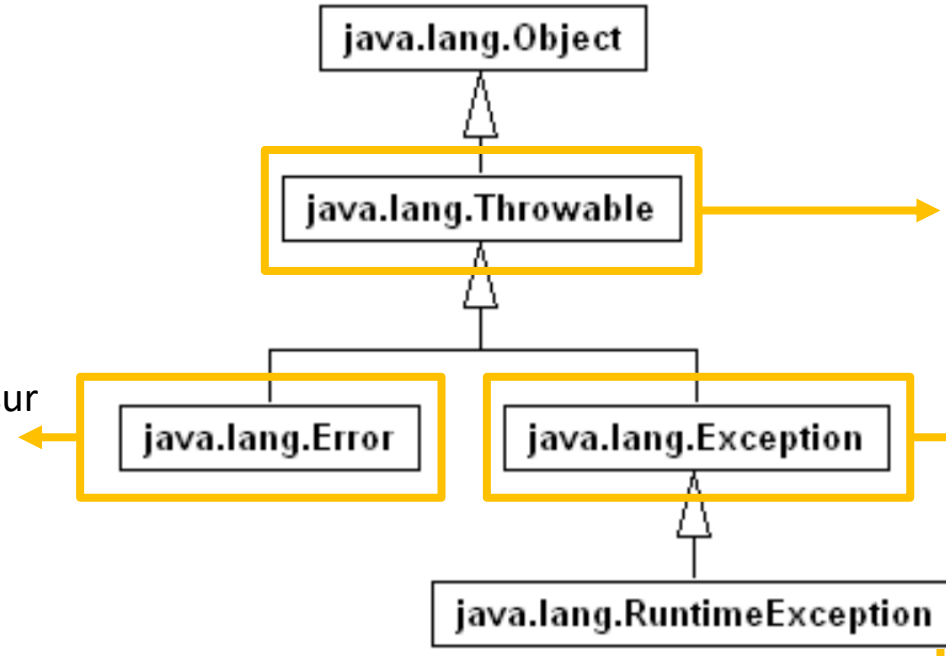
Délégation de la gestion de l'exception

```
public static void main(String[] args) {
   Process p=new Process();
float result;
try{
       result=p.divisionA(45, 0);
   }catch (ArithmeticException e) {
     System.out.println("e:"+e.getMessage());
   }
 }
```

Gestion des exceptions

- Création d'une exception

Extension de la classe **Exception**.



Classe de base pour le traitement des exceptions

Classe d'erreur critique sur la machine virtuelle
→ arrêt de l'application

Classe d'erreur moins critique pouvant être gérée

Exception ne nécessitant pas nécessairement de try catch

Gestion des exceptions

- Création d'une exception

```
public class BadCharException extends Exception{

private String content;
private String regex;

   public BadCharException() {
      super();
   }

   public BadCharException(String content,String regex) {
      super();
      this.content = content;
      this.regex = regex;
   }

   @Override
   public String getMessage() {
      return "Caracteres interdits detectés, attention tentative"
                                    + " d'attaque possible contenu:"+this.content
                                    +", regex do not match:"+this.regex;
   }

}
```

Gestion des exceptions

- Création d'une exception

```
public class CheckString {
private final String REGEX="^[A-Za-z0-9]+$";
private Pattern pattern = Pattern.compile(REGEX);

public boolean checkString(String content) throws BadCharException{
   Matcher matcher = pattern.matcher(content);
   if (matcher.find()){
      return true;
   }else{
      throw new BadCharException(content,REGEX);
   }
}
}
```

Délégation de la gestion de l'exception

Déclenchement d'une exception (création de l'exception custom

```
public static void main(String[] args) {
   CheckString checker=new CheckString();

   try {
      checker.checkString("aabbcc");
      checker.checkString("aabbcc&éé");
   } catch (BadCharException e) {
      e.printStackTrace();
}}
```

```
com.course.examples.exception.BadCharExce
ption: Caracteres interdits detectés,
attention tentative d'attaque possible
contenu:aabbcc&éé, regex do not
match:^[A-Za-z0-9]+$
at
com.course.examples.exception.CheckString
.checkString(CheckString.java:15)
at
com.course.examples.exception.CheckString
.main(CheckString.java:25)
```

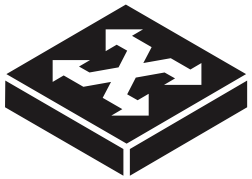
Exercice

Créer 2 Classes exceptions:

- **NoCharException** à déclencher lorsque le contenu = null ou ""
- **NoEmailException** à déclencher lorsque le contenu n'est pas un email

Créer 1 Classe Validation permettant de checker si un champ est null ou vide et si il s'agit bien d'une adresse email (déclencher les exceptions dans les deux cas)

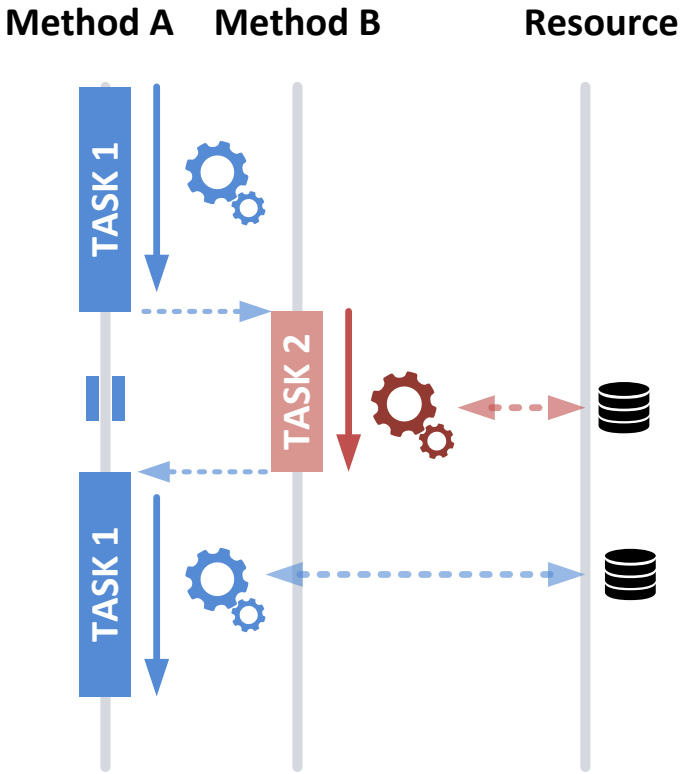
Tester vos classes dans un main



Programmation concurrente de JAVA

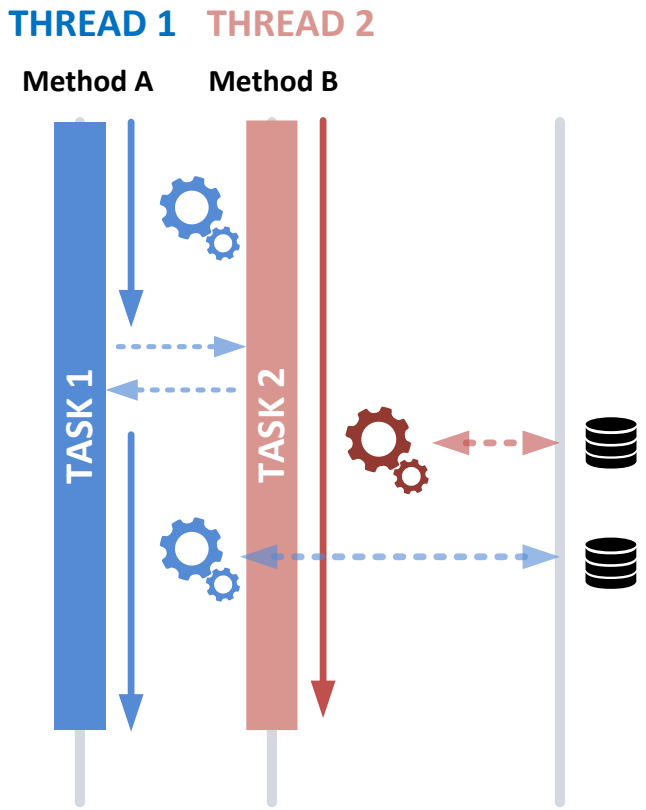
Rappels

1 CPU



Séquentiel

n CPU

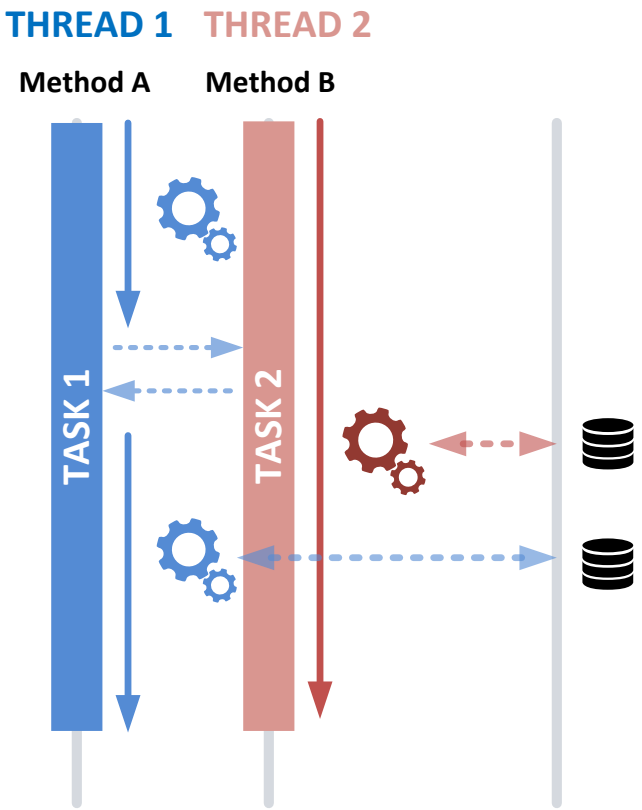


Parallèle

Rappels

Avantages

n CPU



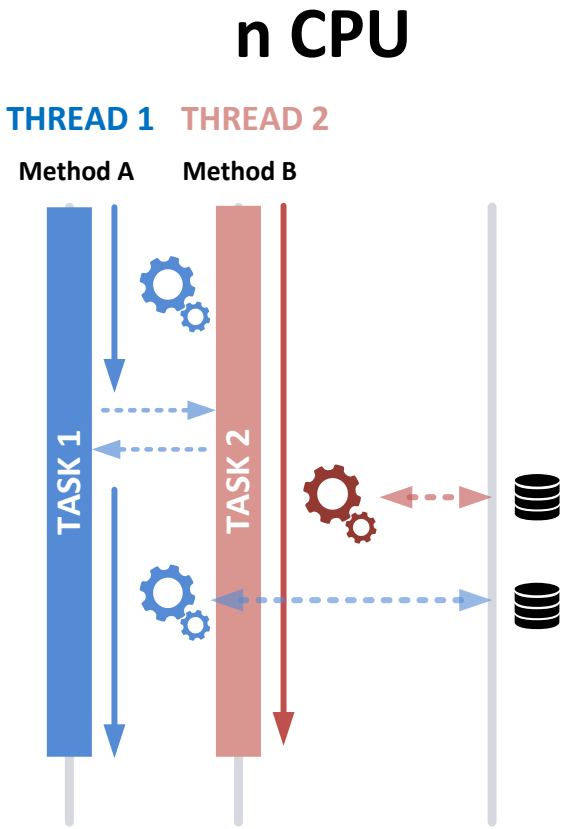
Inconvénients

Parallèle

Rappels

Avantages

- Gain de temps (traitement en parallèle)
- Usage optimal des multi-core CPU
- Améliore la réactivité (GUI)
- Adapté aux problématiques réelles (plusieurs taches exécutées en même temps)



Inconvénients

- Accès concurrents aux ressources
- Interblocages possibles (deadlock)
- Difficile à programmer
- Difficile à débuguer
- Interactions entre les threads complexes
- Résultats difficilement prévisibles

Rappels

- 2 Unités de d'exécution, Processus et Threads
- Processus
  - Possède son propre environnement d'exécution (propre espace de mémoire)
  - Un process peut être vu comme une application ou un programme
  - Les communications entre les processus sont possibles au travers de Pipes ou Socket (Inter Process Communications IPC)
- Threads
  - Peuvent être considérés comme des **processus légers**
  - **1 Thread** ne peut exister qu'à **l'intérieur d'un Process**
  - Les Threads **partagent les ressources un Process** père (mémoires, fichiers ouverts..)
  - Ce **partage** de ressources permet **d'optimiser** les performances (communications) des Threads
  - Une application = au moins 1 Thread (**main Thread**)

Java Concurrency: Class Thread

- Définition

Classe Java permettant de créer un nouveau Thread

- Propriété
  - Peut être contrôlé directement (instanciation manuelle de la classe Thread)
  - Peut être géré par un manager de Thread (**executor**)
  - Exécute une méthode **run()**
  - Doit être démarré explicitement **start()**
  - Un autre processus peut attendre la fin du traitement du thread courant **join()**

Java Concurrency: Class Thread

- Exemple de création de Thread (1/2)

```
public class SimpleThread extends Thread {

   public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
       SimpleThread simpleThread=new SimpleThread();
       simpleThread.start();
    }
}
```

Création d'une classe qui étend Thread

Définition de la méthode d'exécution du Thread

Création du Thread

Démarrage du Thread

- Implémentation la plus simple d'un Thread
- Meilleur lecture du code
- Faible pouvoir de généralisation

Java Concurrency: Class Thread

- Exemple de création de Thread (2/2)

```
public class SimpleRunnable implements Runnable {
    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        Thread myThread=new Thread(new SimpleRunnable());
        myThread.start();
    }
}
```

Création d'une classe qui étend Runnable

Définition de la méthode à exécuter par un Thread

Création d'un Thread avec en paramètre une Classe de type Runnable (méthode à exécuter par le Thread)

Démarrage du Thread

- Plus complexe à mettre en oeuvre
- Fort pouvoir de généralisation

Interactions avec les Threads

- Sleep

Permet du suspendre l'exécution du Thread pour une période donnée et de libérer du temps CPU pour les autres applications/Threads

- Interrupt

Permet d'indiquer au Thread qu'il doit s'arrêter. Le Thread devra détecter cette interruption au travers d'un exception **InterruptedException**

- Join

Permet d'attendre qu'un **Thread** finisse son exécution. Un timer peut être passé en paramètre et déclencher un **interrupt** à la fin de ce dernier

Interactions avec les Threads

- Sleep

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

   for(int i=0;i<100;i++){

      System.out.println("Timer :"+i+"s");

      Thread.sleep(1000);
   }
}
```

Interactions avec les Threads

- Interrupt

```
public class ThreadInteractions extends Thread {
   public void run() {
      try {
          for (int i = 0; i < 100; i++) {
             Thread.sleep(1000);
             System.out.println("Timer :" + i + "s");
          }
        } catch (InterruptedException e) {
           e.printStackTrace();    }}

public static void main(String[] args) throws
                                  InterruptedException {
   ThreadInteractions t=new ThreadInteractions();

   t.start();

   Thread.sleep(5000);

   t.interrupt();
  }
}
```

Résultat

```
Timer :0s
Timer :1s
Timer :2s
Timer :3s
java.lang.InterruptedExcept
ion: sleep interrupted
At java.lang.Thread.sleep(
Native Method)
at
com.course.examples.threads
.ThreadInteractions.run(
ThreadInteractions.java:7)
```

Interactions avec les Threads

- join

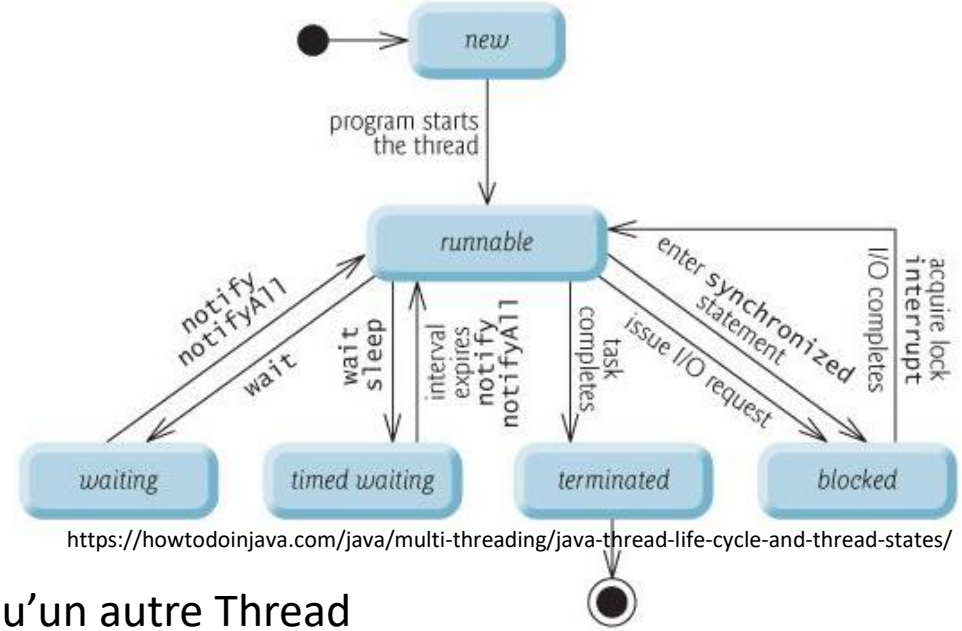
```
public class ThreadInteractions extends Thread {
   public void run() {
      try {
          for (int i = 0; i < 100; i++) {
             Thread.sleep(1000);
             System.out.println("Timer :" + i + "s");
          }
        } catch (InterruptedException e) {
           e.printStackTrace();    }}

 public static void main(String[] args) throws
                                 InterruptedException {

    ThreadInteractions t=new ThreadInteractions();

    t.start();

    t.join();

   }
}
```

Résultat

```
Timer :0s
Timer :1s
Timer :2s
Timer :3s
Timer :4s
Timer :5s
…
Timer :96s
Timer :97s
Timer :98s
Timer :99s
```

Interactions avec les Threads

- Cycle de vie
  - **NEW**
    Thread non démarré
  - **RUNNABLE**
    Thread exécute par la JVM
  - **BLOCKED**
    Thread bloqué attendant la libération d'un lock



https://howtodoinjava.com/java/multi-threading/java-thread-life-cycle-and-thread-states/

  - **WAITING**
    Thread en attente (indéfiniment) qu'un autre Thread exécute une action particulière
  - **TIMED_WAITING**
    Idem Waiting avec un timer
  - **TERMINATED**
    Thread a terminé sa tache

L'état du Thread est accessible via les méthodes **getState()**, **interrupted()**, **isAlive()** …

Exercice

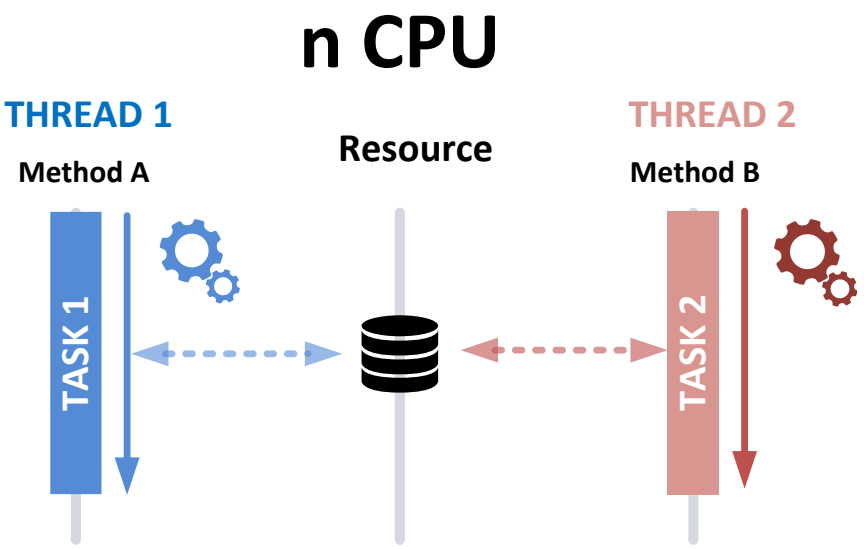
Créer 2 Classes Runnable:

- TimeRunnable permettant d'afficher le tps écoulé depuis le démarrage (pendant 10s)
- PongRunnable permettant d'afficher indéfiniment Pong et Ping alternativement (toutes les secondes)

Créer 1 ExoMain exécutant (dans un main)

- **Créer 2 Threads** utilisant respectivement TimeRunnable, PongRunnable
- **Démarrer** ces 2 Threads et **attendre** que le **Thread 1 (TimeRunnable)** finisse
- **Interrompre** le **Thread 2 (PongRunnable)**

Que se passe-t-il lorsque plusieurs Threads accèdent une même ressource en même Temps ?



Concurrency Exception

- Accès simultanée à la même ressources
  - **Le traitement en cours du Thread 1 peut être détruit par le Thread 2**
  - **La donnée récupérée par le Thread 1 peut changer pendant l'opération effectuée (c=c+1)**
  - **Le traitement du Thread 1 et du Thread 2 peut s'effectuer normalement**
  - **L'accès simultané à la ressource en écriture peut engendrer une Erreur**

Le résultat d'un accès simultané à une ressource est imprévisible

Protéger l'accès simultané aux données

- Utilisation du *Synchronized*
  - **Synchronized method:** ajouter le mot clé *Synchronized* à la méthode cible
    - Interdit l'appel simultané de la méthode cible (suspension de l'exécution du Thread si méthode déjà est déjà appelée)
    - Les changements effectués sur l'objet sont visibles des autres Thread (notamment ceux suspendus)

```
public class BankAccount {
   private float content = 0;

   public synchronized float getContent() {
      return content;
   }

   public synchronized void setContent(float content) {
      this.content = content;
   }
}
```

Protéger l'accès simultané aux données

- Utilisation du *Synchronized Statement*
  - Utilisation d'un bloc empêchant d'autres Threads d'accéder aux traitements du bloc (permet de cibler les zones de code à synchroniser)
  - Utilisation d'un objet pour verrouiller l'accès au bloc de traitement

```
public class BankAccount {
   private float content = 0;
   private Object lockObj=new Object();

   public boolean outcome(float s) {

      synchronized(lockObj){
         if (content - s > 0) {
            content=content-s;
            return true;
         } else {
            return false;
         }
      }
}
```

Object sur lequel s'effectue le verrouillage

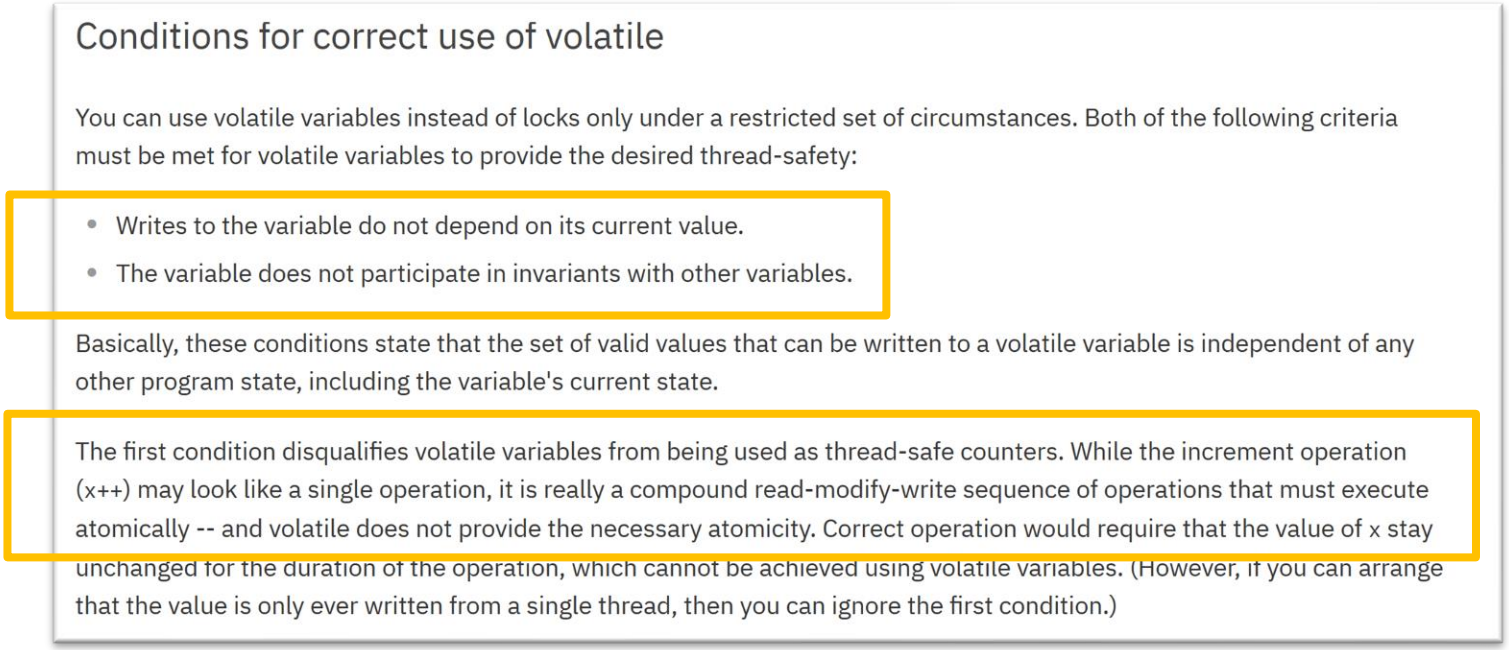
Bloc de traitement accessible à un seul Thread à la fois (libération du lock à la fin du bloc)

Protéger l'accès simultané aux données

- *Atomic Access*
  - Une action atomic ne peut pas être interrompue
    - Actions de Lecture /écriture pour des variables de référence ou pour des types primitifs (int, float,…)
    - Actions de Lecture /écriture pour toutes les variables déclarées *volatile*
  - Les actions atomiques ne peuvent pas être interrompues mais doivent être synchronisées si besoin.
  - L'usage de *volatile* réduit le risque d'erreur en mémoire (consistance)
  - Usage de *volatile* est plus efficace que d'accéder aux ressources via des méthodes synchronisées (attention néanmoins à la coordination des valeurs)
  - Les différents Thread souhaitant accéder à la donnée auront la valeur de la variable volatile la plus à jour

Protéger l'accès simultané aux données

- *Atomic Access*



Conditions for correct use of volatile

You can use volatile variables instead of locks only under a restricted set of circumstances. Both of the following criteria must be met for volatile variables to provide the desired thread-safety:

- Writes to the variable do not depend on its current value.
- The variable does not participate in invariants with other variables.

Basically, these conditions state that the set of valid values that can be written to a volatile variable is independent of any other program state, including the variable's current state.

The first condition disqualifies volatile variables from being used as thread-safe counters. While the increment operation (x++) may look like a single operation, it is really a compound read-modify-write sequence of operations that must execute atomically -- and volatile does not provide the necessary atomicity. Correct operation would require that the value of x stay unchanged for the duration of the operation, which cannot be achieved using volatile variables. (However, if you can arrange that the value is only ever written from a single thread, then you can ignore the first condition.)

https://www.ibm.com/developerworks/java/library/j-jtp06197/index.html

Protéger l'accès simultané aux données

- *Atomic Access*

```
public class UserManager {
    public volatile String lastUser;

    public boolean authenticate(String user, String password) {
        boolean valid = passwordIsValid(user, password);
        if (valid) {
            User u = new User();
            activeUsers.add(u);
            lastUser = user;
        }
        return valid;
    }
}
```

Protéger l'accès simultané aux données

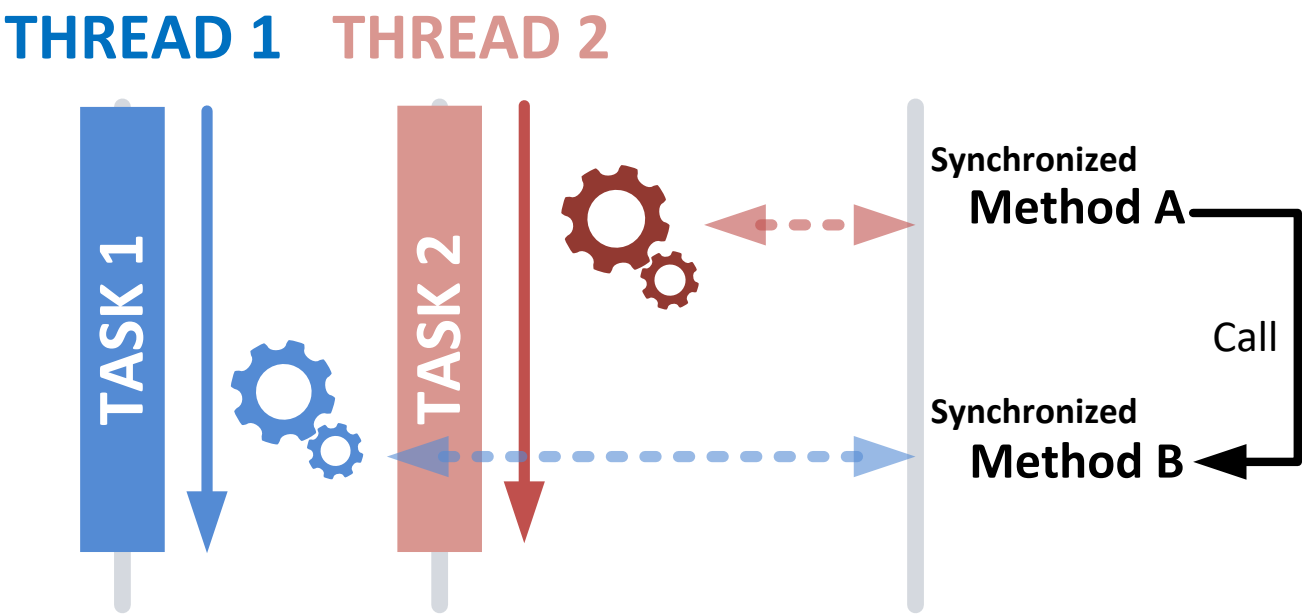
- *Immutable Object*
  - Déclaration d'objets qui ne pourront pas être modifiés une fois instanciés
    - Utilisation du mot clé ***final*** pour définir un objet immutable.
    - Si l'objet immutable fait référence à d'autres objets s'assurer que ces derniers ne seront pas modifiés
    - *Les objets immutable déclarés à l'aide de **final** pourront être accédés simultanément pas plusieurs Threads*

```
public class Account {

   final private float rate;

   private float value;

   public Account(float rate) {
      this.rate=rate;
   }

   public float getRate() {
      return rate;
   }

   public synchronized float
                     getValue() {…}
   public void synchronized
          setValue(float value) {…}

}
```

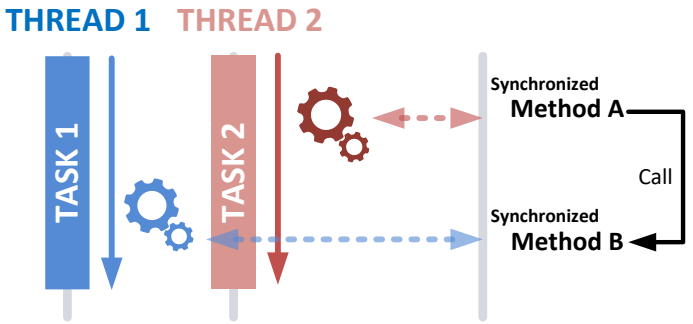
Les autres dangers du multi-Threading

- **Dead-Lock**
- **Live-Lock**
- **Starvation**

Le Dead Lock !

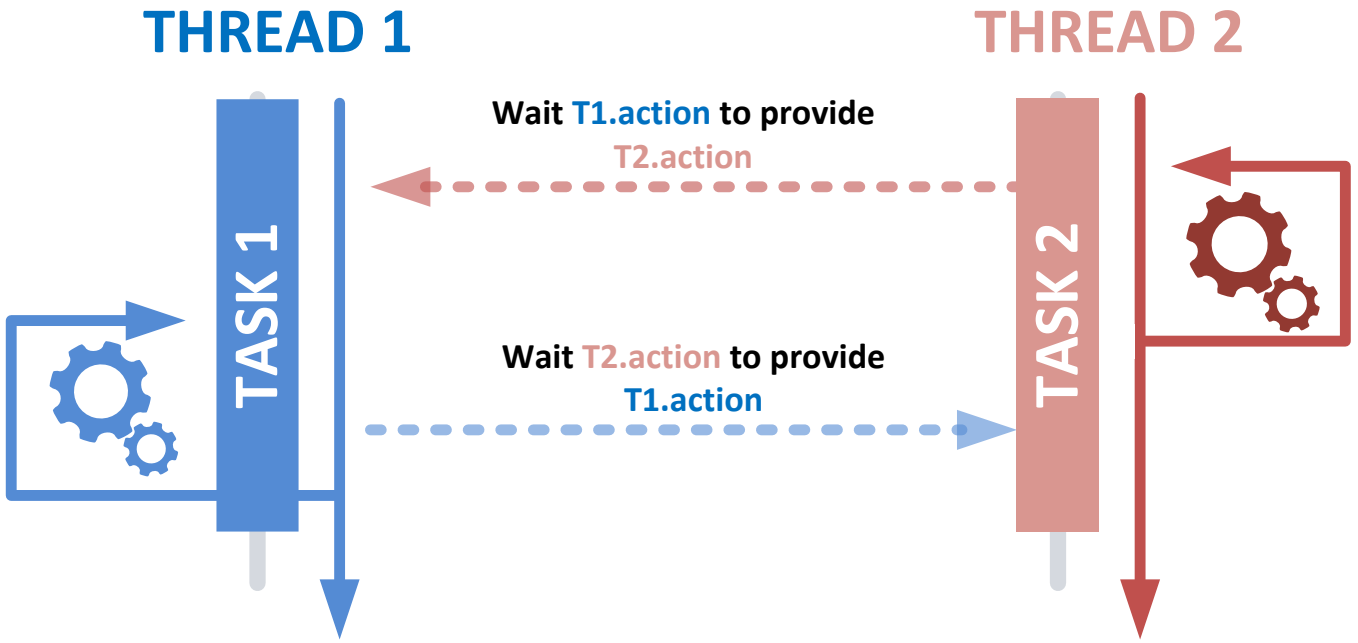


Le Dead Lock !

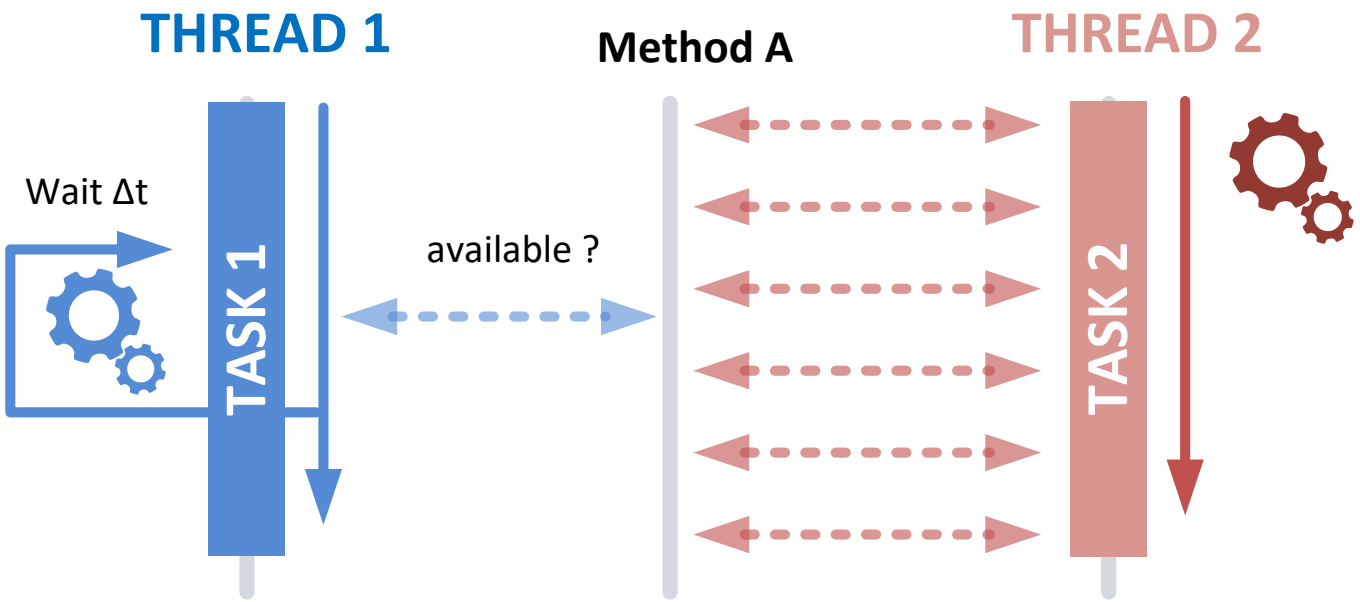



```
public class Counter {
   private float counter=0;


   public synchronized float getCounter() {
      return counter;
   }

   public synchronized void setCounter(float counter) {
      this.counter = counter;
      System.out.println(getCounter());
   }

}
```

Le Live Lock !



Le Starvation !



Synchronisation des actions

```
public class BlockingQueue<T> {
    private Queue<T> queue = new LinkedList<T>();
    private int capacity;

    public BlockingQueue(int capacity) {
        this.capacity = capacity;
    }

    public synchronized void put(T element)
                    throws InterruptedException {
        while(queue.size() == capacity) {
            wait();
        }
        queue.add(element);
        notifyAll();
    }
    public synchronized T take()
                    throws InterruptedException {
        while(queue.isEmpty()) {
            wait();
        }
        T item = queue.remove();
        notifyAll() ;
        return item;
    }
```

- *Guard Block*

  Block de traitement d'un Thread en attente du résultat d'un autre Thread
  Utiliser les mots clé **Wait** et **notifyAll**

Programmation concurrente de JAVA : Objets avancés

Gestion de la concurrence : Objets Avancés

- ***Lock***

  *Objet Avancé permettant d'éviter l'accès concurrent à un bloc de code (comme **synchronized)** et supportant les actions de **wait** et **notify***

- ***Executor***

  Utilitaire permettant l'exécution de la gestion de multiples Threads

- ***Concurent Collection***

  Ensemble d'utilitaires de collection permettant un usage concurrent des List, Map et Set

Gestion de la concurrence : Lock

- *Fonctionnalités:*
  - *Création d'un jeton* — **Lock lock = new ReentrantLock();**
  - *Réservation de jeton* — **lock.lock()**
  - *Libération de jeton* — **lock.unlock()**
  - *Réservation du jeton que si non réserve* — **lock.tryLock()**
  - *Tentative de réservation de jeton avec un timeout* — **lock.tryLock(100L, TimeUnit.*MILLISECONDS*)**
  - *Permet d'attendre des conditions avant de continuer* — ***Condition myC = lock.newCondition();*** ***myC.await();*** ***myC.signalAll();***

Gestion de la concurrence : Lock

- ***E.g ReentrantLock***
  - *Implémente l'interface* ***Lock***
  - *Implémentation des méthodes similaires au mot clé* ***Synchronized***
  - *Possibilité de récupérer le Thread possédant le jeton, les Threads en attente, etc..*

Gestion de la concurrence : Lock

```
public class TestLock {
   private Lock lock = new ReentrantLock();
   private Condition maintenanceC=lock.newCondition();
   private boolean isInMaintenance=false;
   private float value;

   public TestLock() { }

   public void setValue(float value) throws InterruptedException {
   if(isInMaintenance){
      maintenanceC.await();
   }
   try{
      lock.tryLock(100L, TimeUnit.MILLISECONDS);
      this.value = value;
   }finally {
      lock.unlock();
   }
}

public void objectMaintenance() throws InterruptedException{
   isInMaintenance=true;
   Thread.sleep(5000);
   isInMaintenance=false;
   maintenanceC.notifyAll();
}}
```

Gestion de la concurrence : Executor

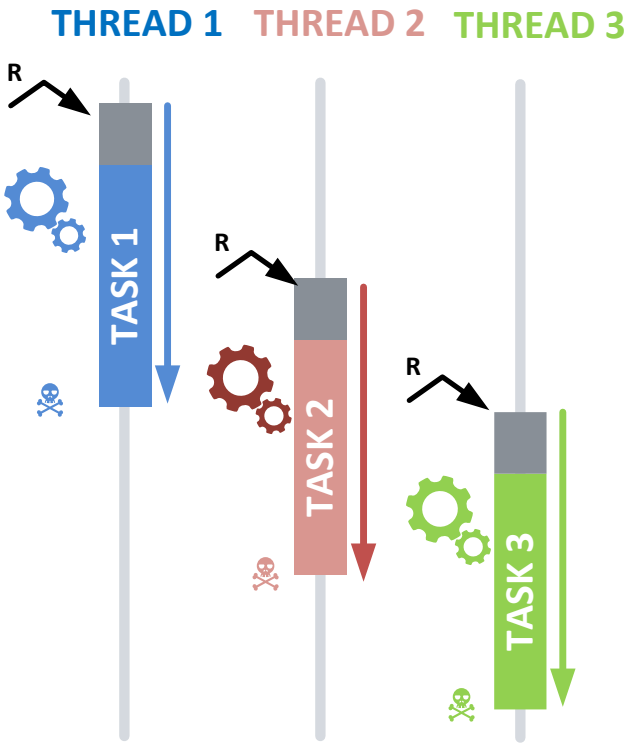
- ***Définition***

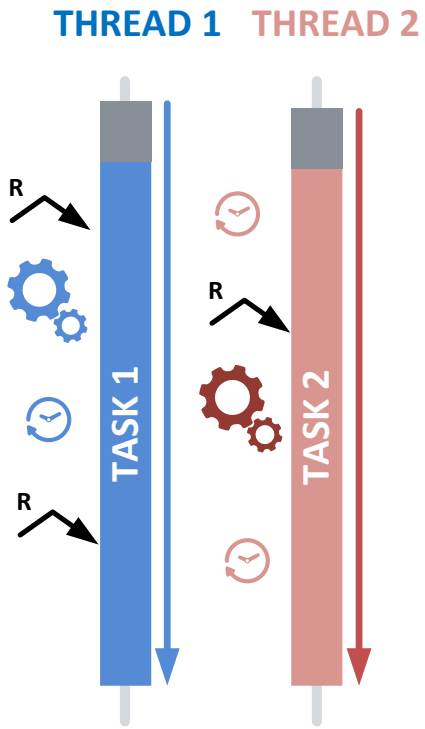
Ensemble d'utilitaires permettant la gestion (en masse) de Thread et ainsi d'optimiser les ressources (optimisation de la création de Thread → Working Thread) et de manager le cycle de vie des Threads

- ***Organisation: les Interfaces Executor***
  - ***Executor**: fournit une méthode **execute()** permettant d'exéctuer des classes **Runnable***
  - ***ExecutorService**: extension de **Executor** propose une méthode **submit()** proposant les fonctionnalités identique à **execute()** mais acceptant les Classes **Callable** (Runnable avec une valeur retour possible)*
  - ***ScheduledExecutorService**: extension de ExecutorService proposant des utilitaires supplémentaires permettant de **planifier** des exécutions de **Runnable** ou **Callable***

Gestion de la concurrence : Executor

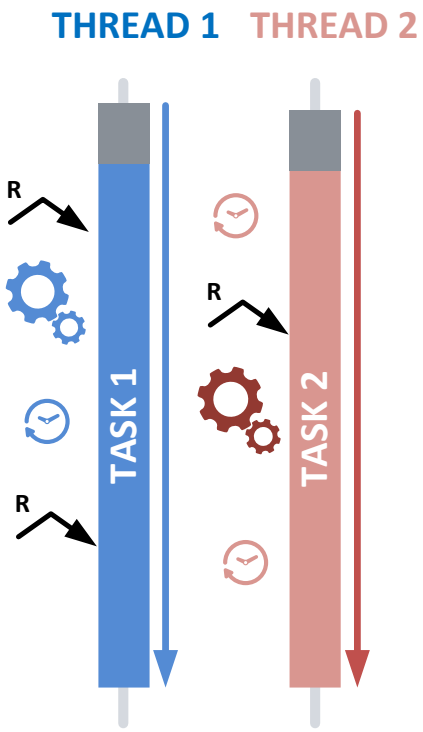
- ***Fonctionnement, les Threads Pool***





Gestion de la concurrence : Executor

- ***Fonctionnement, les Threads Pool***
  - ***Cached thread pool***
    Mise en place d'un certain nombre de Thread et création de Threads supplémentaires si besoin
  - ***Fixed thread pool***
    Limite le nombre maximum de Threads concurrents (les jobs additionnels sont mis en attente dans une Queue)
  - ***Single-threads pool:***
    1 seul Thread dans le pool (les jobs additionnels sont mis en attente dans une Queue)
  - ***Fork/Join Pool :***
    Utilisation du Framework Fork/join permettant de diviser les tâches complexe (traitement plus long) en taches plus petites (récursives)



Gestion de la concurrence : Executor

```
public class SimpleExecutorExample {

    public static void main(String[] args) {
        ExecutorService pool =
                Executors.newSingleThreadExecutor();

        Runnable task = new Runnable() {
            public void run() {
                System.out.println(
                   Thread.currentThread().getName()
                   );
            }
        };
        pool.execute(task);
        pool.shutdown();
    }
}
```

Exemple de code de www.codejava.net

Gestion de la concurrence : Executor

```
public class SimpleExecutorServiceExample {
    public static void main(String[] args) {
        ExecutorService pool = Executors.newSingleThreadExecutor();
        Callable<Integer> task = new Callable<Integer>() {
            public Integer call() {
                try {
                    // fake computation time
                    Thread.sleep(5000);
                } catch (InterruptedException ex) {
                    ex.printStackTrace();
                }
                return 1000;
            }
        };
        Future<Integer> result = pool.submit(task);
        try {
            Integer returnValue = result.get();
            System.out.println("Return value = " + returnValue);

        } catch (InterruptedException | ExecutionException ex) {
            ex.printStackTrace();
        }
        pool.shutdown();
    }
}
```

Exemple de code de www.codejava.net

Gestion de la concurrence : Executor

```
public class CountDownClock extends Thread {
    private String clockName;
    public CountDownClock(String clockName) {
        this.clockName = clockName;
    }
    public void run() {
        String threadName = Thread.currentThread().getName();
        for (int i = 5; i >= 0; i--) {
            System.out.printf("%s -> %s: %d\n",
                                    threadName, clockName, i);
            try {
                Thread.sleep(1000);
            } catch (InterruptedException ex) {
                ex.printStackTrace();
            }
        }
    }
}

public class MultipleTasksExecutorExample {
    public static void main(String[] args) {
        ExecutorService pool = Executors.newCachedThreadPool();
        pool.execute(new CountDownClock("A"));
        pool.execute(new CountDownClock("B"));
        pool.execute(new CountDownClock("C"));
        pool.shutdown();   }}
```

Exemple de code de www.codejava.net

Gestion de la concurrence : Collection

- ***Définition***

Ensemble d'utilitaires permettant d'éviter les accès concurrents à la mémoire.

Création à partir des collections existantes

- ***Interfaces Utilitaires***
  - ***BlockingQueue:*** *FIFO avec attente bloquante ou time out*
  - ***ConcurrentMap:*** *définie des opérations atomic. Implémentation la plus courante* ***ConcurrentHashMap****.*
  - ***ConcurrentNavigableMap****: sous interface de concurentMap supportant des « matches » approximatif. Implémentation la plus courante* ***ConcurrentSkipListMap***

Exercice

Executor

- Créer une classe **SynchonizedCounter** permettant de modifier un int par plusieurs Thread en même temps
- Créer une classe **UpdateCounter** (class Runnable) ayant en paramètres
  - le tps de sleep
  - L'objet SynchonizedCounter à mettre à jour.
  - La valeur d'incrément

  Cette classe mettra à jour le counter avec sa valeur d'incrément toutes les x secondes
- Créer une Class Launch possédant un main permettant
- Créer un **Executor** exécutant 4 **UpdateCounter** possédant des paramètre différents
- Affichant toutes les secondes la valeur du compteur

I/O Gestion des fichiers avancées

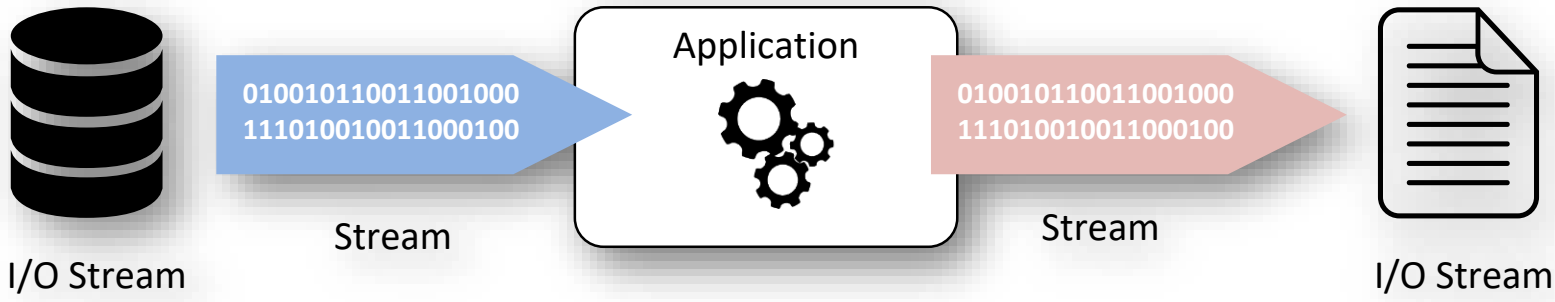
Gestion des fichiers

- ***Les Streams***

Suite de données (finies ou non) gérée de façon temporelle

- ***Les I/O Streams***

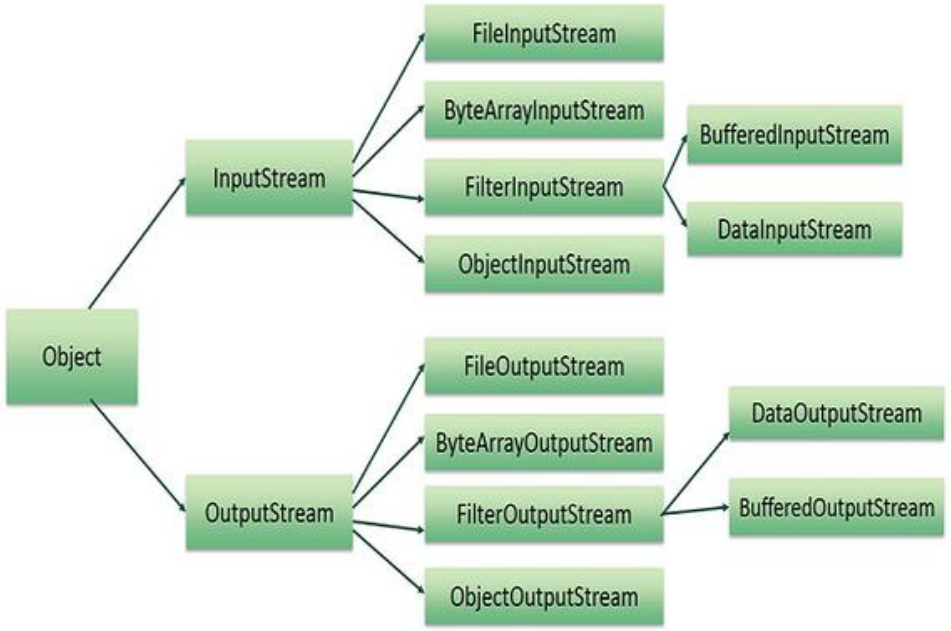
sources d'information ou réceptacles de données. Ils peuvent être de différentes natures: fichier, device, des programmes des mémoires etc…



Gestion des fichiers

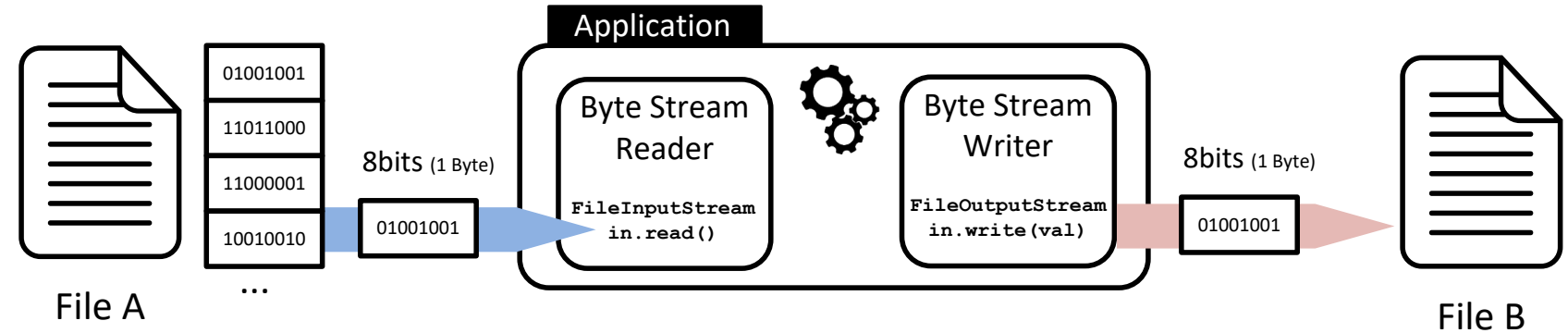
- ***Types de Streams***
  - *InputStream : utilisé pour lire l'information provenant d'une source de données*
  - *OutputStream : utilisé pour écrire une information dans une destination*

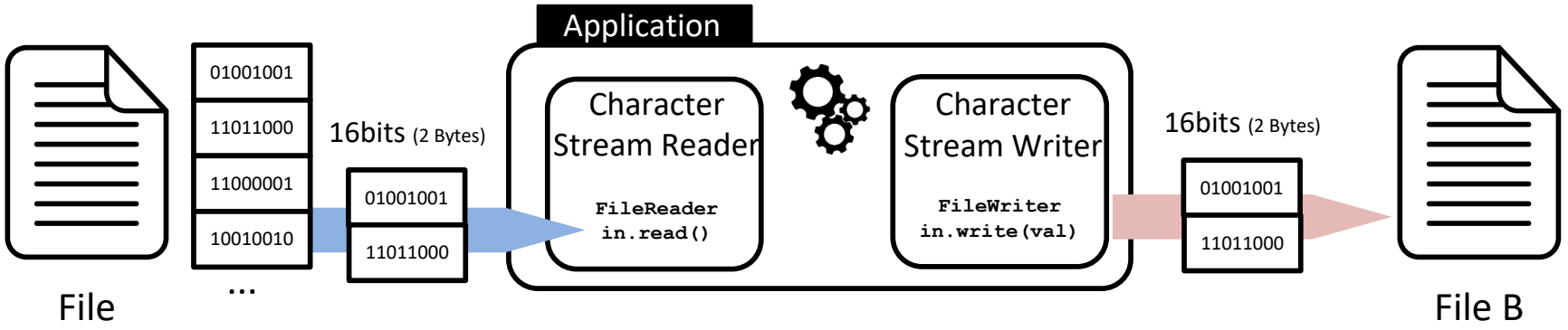
- ***Les différents formats de Stream (fondamentaux)***
  - *Byte Streams: input et ouput d'octet de 8 bits (e.g* **FileInputStream** and **FileOutputStream***)*
  - *Characters Streams: input et output pour du 16-bit unicode (e.g* **FileReader** and **FileWriter***)*



http://www.tutorialspoint.com/java/java_files_io.htm

Gestion des fichiers





Gestion des fichiers

```
public class CopyBytesFile {
    public static void main(String[] args)
                            throws IOException {
        FileInputStream in = null;
        FileOutputStream out = null;
        try {
            String path="./src/com/course/examples/io/";
            in = new FileInputStream(path+"test.txt");
            out = new FileOutputStream(path+"output.txt");
            int c;

            while ((c = in.read()) != -1) {
                out.write(c);
            }
        } finally {
            if (in != null) {
                in.close();
            }
            if (out != null) {
                out.close();
            }
        }
    }
}
```

Toujours fermer les Flux !!

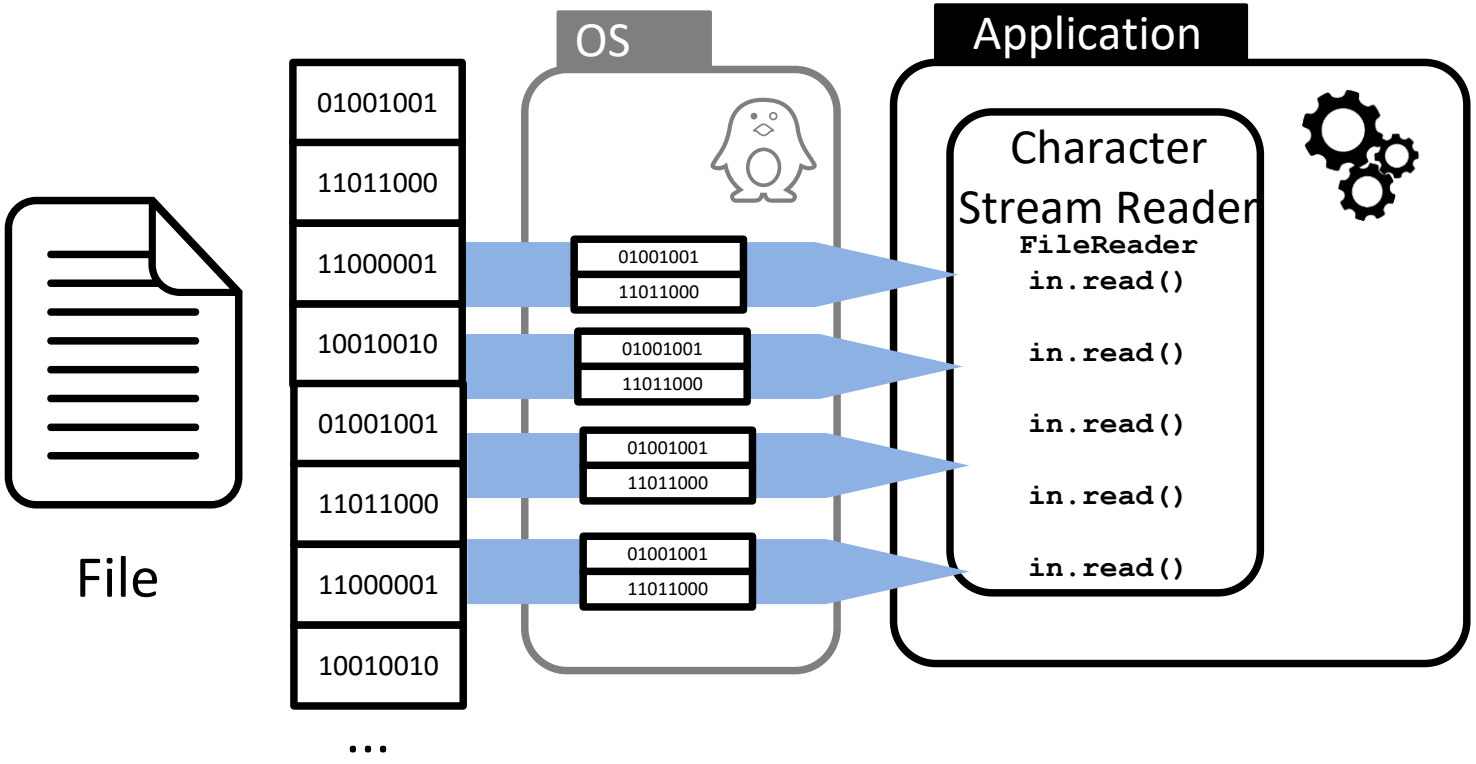
Gestion des fichiers

```
public class ChangeFileChar {
  public static void main(String[] args) throws IOException
{
        FileReader inputStream = null;
        FileWriter outputStream = null;
        try {
            String path="./src/com/course/examples/io/";

            inputStream = new FileReader(path+"char1.txt");
            outputStream = new FileWriter(path+"char2.txt");

            char c;
            while ((c = (char) inputStream.read()) != -1) {
            if( c == 'e'){
               c='E';
            }
                outputStream.write(c);
            }
        } finally {
            if (inputStream != null) {
                inputStream.close();
            }
            if (outputStream != null) {
                outputStream.close();
            } } } }
```

Toujours fermer les Flux !!

Gestion des fichiers

- ***Optimisation***

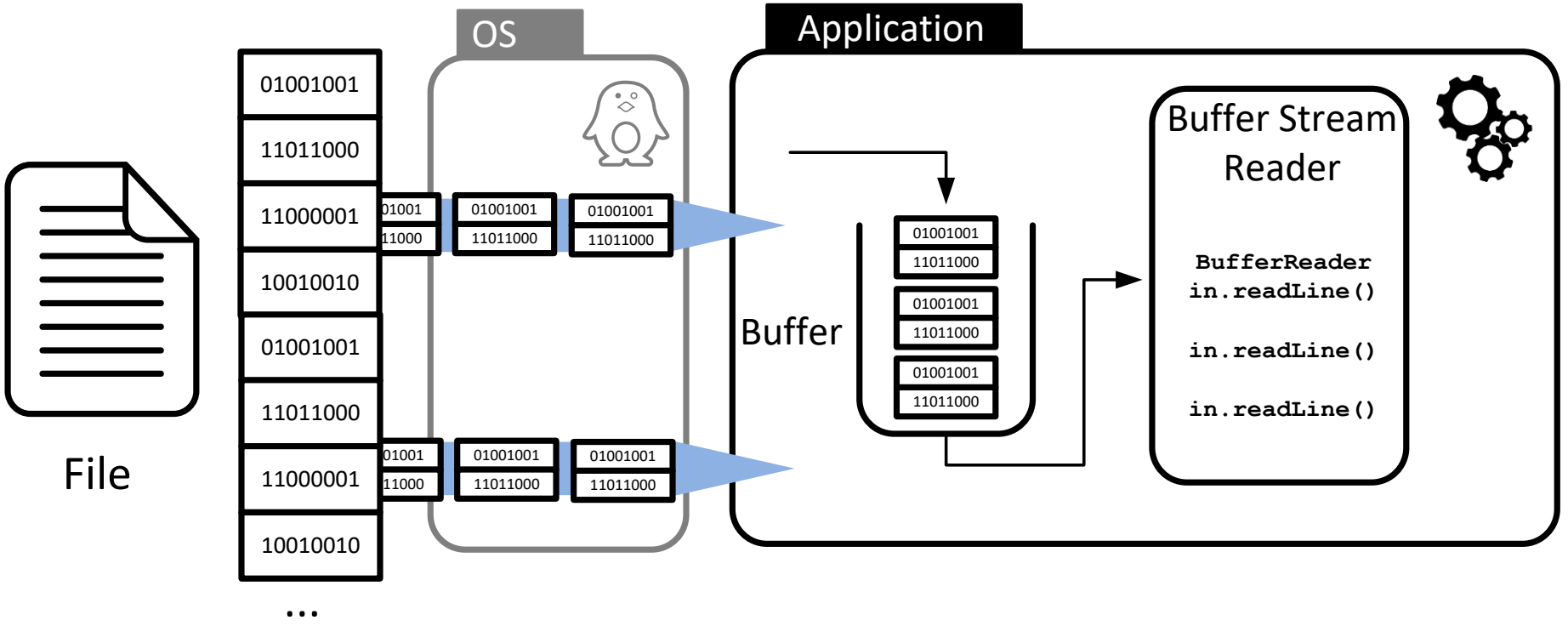


Chaque requête de lecture et d'écriture est traitée par l'OS

→ beaucoup d'accès disque , réseaux →perte de performance

Gestion des fichiers

- ***Optimisation***



Appel de la lecture uniquement lorsque le Buffer est vide
Appel de l'écriture uniquement lorsque le Buffer est plein → ATTENTION PENSER au FLUSH !!

Gestion des fichiers

```
public class ChangeFileBufferChar {
  public static void main(String[] args) throws IOException {
  BufferedReader inputStream = null;
  PrintWriter outputStream = null;
        try {
        String path="./src/com/course/examples/io/";

            inputStream = new BufferedReader(new FileReader(path+"char1.txt"));
            outputStream = new PrintWriter(new FileWriter(path+"char2.txt"));

            String l;
            while ((l = inputStream.readLine()) != null) {
            String lNew=l.replace('t','f');
                outputStream.println(lNew);
            }
        } finally {
            if (inputStream != null) {
                inputStream.close();
            }
            if (outputStream != null) {
                outputStream.flush();
                outputStream.close();
            }
        }
    }
}
```

Gestion des fichiers

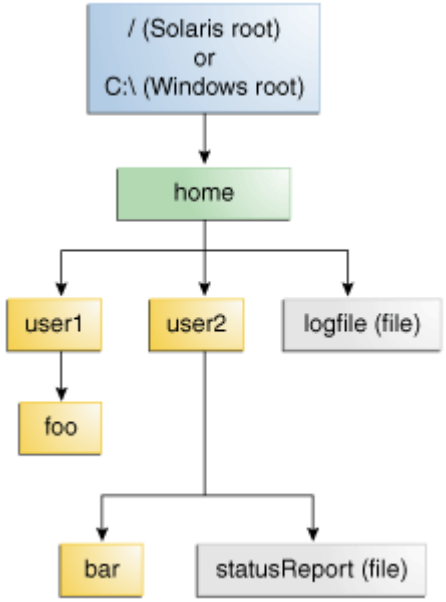
- ***Manipulation de Dossiers et Fichiers***
  - ***La Classe Paths***

    Classe utilitaire permettant d'interagir avec un dossier (trouver, rechercher, parcourir).
    Cette classe dépend de l'OS (e.g '/var/log' ou 'c:\var\log')
  - ***La Classe Files***

    Permet de manipuler les dossiers et les fichiers (existence, création suppression, info)

PATH

```
p1=Paths.get("/home/user2")
      p1.getRoot()
   p1.toAbsolutePath()
for (Path name: path) {}
```



Files

```
Files.delete("/home/user2")
  Files.exists(Path p)
Files.getLastModifiedTime
  ("/home/user2")
```

Gestion des fichiers

- ***Manipulation de Dossiers et Fichiers : la classe PATHS***
  - *Fonctionnalités*
    - *Chargement d'un dossier* `Path p1=Paths.get('/var/log')`
    - *Information sur un dossier (nom, sous chemin, parent , Root)*
    - *Conversion du chemin (*`toUri()`, `toAbsolutePath()`, `toRealPath()`*)*
    - *Comparer deux chemins (*`equal()`, `startsWith()`, `endsWith()`*)*
    - *Lister le contenu d'un chemin (*`Iterable`*)*
    - *Convertir le* ***Path*** *en objet* ***File***

JAVADOC de la classe Path complète disponible ici:
https://docs.oracle.com/javase/7/docs/api/java/nio/file/Path.html

Gestion des fichiers

- ***Manipulation de Dossiers et Fichiers : la classe Files***
  - *Fonctionnalités*
    - *Création de **Stream** sur un **Path***
    - *Déplacement de Fichiers/Dossiers* `( Files.move(Path p1,Path p2… ) )`
    - *Copie d'un Fichier/Dossier* `(Files.copy(source, target, REPLACE_EXISTING))`
    - *Vérification de l'existence* `( Files.exists(path) )`
    - *Vérification des droits (*`isReadable(Path p)`*,* `isWritable(Path p),…`*)*
    - *Lecture des informations du Fichier/dossier (*`isDirectory(Path p),`
      `size(Path p),getLastModifiedTime()`*)*

Voir le lien ci-dessous pour les informations détaillés des métaData
https://docs.oracle.com/javase/tutorial/essential/io/fileAttr.html

Gestion des fichiers

- ***Manipulation de Dossiers et Fichiers : la classe File***

Représentation d'un objet fichier ou dossier spécifique, API d'opération de création suppression, update, récupération d'information.

```
Path p2 = Paths.get("./src/com/course/examples/io/");

System.out.println("full Path: "+p2.toAbsolutePath());

for(File f :p2.toFile().listFiles()){
   String prefix="-";

   if(f.isDirectory()){
   prefix="D";
}
```

JAVADOC complète disponible ici:
https://docs.oracle.com/javase/7/docs/api/java/io/File.html

Gestion des fichiers

```
public class PathFilesSample {

public static void main(String[] args) throws IOException {
   Path p1 = Paths.get("./src/com/course/examples/io/test.txt");
   System.out.println("full Path: "+p1.toAbsolutePath());

   BasicFileAttributes att = Files.readAttributes(p1, BasicFileAttributes.class);
   System.out.println("   "+p1+"-   "+att.creationTime()+"   -"+att.lastModifiedTime());

   Path p2 = Paths.get("./src/com/course/examples/io/");
   System.out.println("full Path: "+p2.toAbsolutePath());
   for(File f :p2.toFile().listFiles()){
      String prefix="-";

      if(f.isDirectory()){
         prefix="D";
      }

      BasicFileAttributes att2 = Files.readAttributes(f.toPath(),
                                                      BasicFileAttributes.class);
      System.out.println(prefix+"-"+f.getPath()+"-"+att2.creationTime()+
                                                      "-"+f.lastModified());
   }
}
```

Gestion des fichiers

- ***Notions avancée supplémentaires***

  - ***Scanner et Formatting:*** *objet permettant de découper les flux de donnée (Scanner) et de le formater au besoin (Formatting)*

  - ***File Store:*** *utilitaire permettant de récupérer notamment les informations sur l'espace mémoire disponible/total/utilisé etc..*

  - ***File Operation : les Glob, syntaxe permettant de filtrer les éléments recherchés***
    (*[0-9]* – Matches all strings containing a numeric value)

Pour une description complete voir la documentation ORACLE
https://docs.oracle.com/javase/tutorial/essential/io/index.html

Exercice

Créer une Class KeyWordReadFolder permettant:

- Lire le contenu d'un répertoire
- Lister l'ensemble des fichiers de ce répertoire
- Sauvegarder dans une liste l'ensemble des mots des fichiers des répertoires
- Afficher la liste des mots

Exercice

Créer une Class StatWordReadFolder permettant:

- Lire le contenu d'un répertoire
- Lister l'ensemble des fichiers de ce répertoire
- De compter l'occurrence d'un mot clé dans chaque fichier
- De stocker l'occurrence du mot clé par fichier dans une Map
- D'afficher la Map

Exercice

Mettre à jour la Class StatWordReadFolder afin de:

- Sauvegarder les stats de l'ensemble des fichiers dans un fichier unique savStat.txt

Mettre à jour la Class StatWordReadFolder afin de:

- Sauvegarder les stats de chaque fichier dans un fichier séparé e.g savStat-fileName.txt

Exercice

Mettre à jour la Class StatWordReadFolder permettant:

- compter une liste de mots clés par fichiers

Exercice

1 Thread en lecture en boucle sur 1 répertoire, si nouveau fichier chargement des **mots à compter (KeyWordReadFolder)**

1 Thread en lecture en boucle sur 1 répertoire si nouveaux **fichiers** compter les nombres de mots (cf mots à compter) **(StatWordReadFolder)**

1 Thread qui toutes les 20s sauvegarde

- 1 fichier stat. par fichier (nb mots comptés dans fichier)
- 1 fichier stat. global (nb mots comptés tous les fichiers)

Questions ?

References

References

- **Web references**
  - Classes package
    - https://docs.oracle.com/javase/tutorial/java/javaOO/classdecl.html
  - Héritage / interface / classe abstraite
    - https://docs.oracle.com/javase/tutorial/java/concepts/inheritance.html
    - https://docs.oracle.com/javase/tutorial/java/IandI/index.html
    - https://programming.guide/java/clone-and-cloneable.html
    - https://dzone.com/articles/java-interface-vs-abstract-class
    - https://www.javaworld.com/article/2077421/learn-java/abstract-classes-vs-interfaces.html
  - Uml/Diagramme de classe
    - http://users.teilar.gr/~gkakaron/oose/04.pdf
    - https://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/index.html
  - Collection
    - https://www.mainjava.com/java/core-java/complete-collection-framework-in-java-with-programming-example/
    - http://tutorials.jenkov.com/java-collections/index.html
    - http://www.javapractices.com/topic/TopicAction.do?Id=65
  - Générique
    - https://docs.oracle.com/javase/tutorial/java/generics/types.html
  - Exception
    - https://www.jmdoudoux.fr/java/dej/chap-exceptions.htm
  - Concurrence
    - https://docs.oracle.com/javase/tutorial/essential/concurrency/
  - I/O
    - https://docs.oracle.com/javase/tutorial/essential/io/charstreams.html
  - Annotation
    - https://docs.oracle.com/javase/tutorial/java/annotations/index.html
    - https://www.jmdoudoux.fr/java/dej/chap-annotations.htm

References

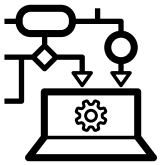
- **Web references**
  - Maven
    - https://maven.apache.org/guides/getting-started/
    - https://java.developpez.com/tutoriels/java/maven-book/
    - http://igm.univ-mlv.fr/~dr/XPOSE2010/Apache_Maven/introduction.html
    - https://mermet.users.greyc.fr/Enseignement/CoursPDF/maven.pdfTests
  - Tests
    - http://www.test-recette.fr/tests-techniques/
  - JVM
    - https://www.geeksforgeeks.org/jvm-works-jvm-architecture/ ->TB
    - https://javatutorial.net/jvm-explained
    - https://www.cubrid.org/blog/understanding-jvm-internals/

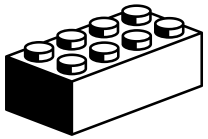
- **Livre / autres ressources**
  - Programmation Orientée Objet en Java, F. Perrin, CPE Lyon
  - Kathy SIERRA et Bert BATES pour leur ouvrage Java -Tête la Première (O'Reilly edition)

References

creative commons

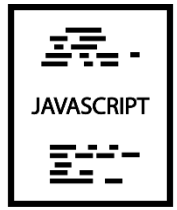

Created by Demograph
from the Noun Project


Created by H Alberto Gongora
from the Noun Project


Created by jon trillana


Created by lastspark
from Noun Project


Created by Aha-Soft
from Noun Project



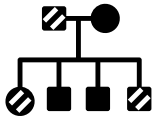
Created by AlfredoCreates.com
from Noun Project


Created by Trần Quang Hiển
from the Noun Project

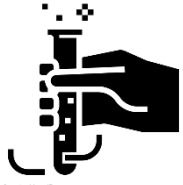

Created by Aldric Rodriguez
from the Noun Project


Created by parkjisun
from the Noun Project

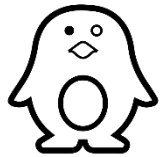

Created by priyanka
from the Noun Project


Created by dDara


Created by Opher Aloni
from the Noun Project


Created by dDara
from Noun Project


Created by Krisada
from Noun Project


Created by Delwar Hossain
from Noun Project


Created by Mohammed Tarequl Alam
from Noun Project


Created by Delwar Hossain
from Noun Project

ÉCOLE SUPÉRIEURE
DE CHIMIE PHYSIQUE ÉLECTRONIQUE
DE LYON

Jacques Saraydaryan

Jacques.saraydaryan@cpe.fr