ITI 1521. Introduction à l'informatique II

Traitement des erreurs en Java

by

Marcel Turcotte

Version du 24 février 2020

Aperçu

Traitement des erreurs en Java

Les langages modernes de programmation offrent des mécanismes pour les traitements d'erreurs. En Java, nous verrons qu'une situation d'erreur est modélisée à l'aide d'un objet. Nous verrons comment signifier une erreur et en comprendrons les conséquences sur le flot de contrôle. Finalement, nous examinons les deux façons de gérer les erreurs.

Objectif général :

- À la suite de cours, vous serez en mesure de signifier et gérer les erreurs en Java.

Objectifs d'apprentissage

- **Nommer** quelques types d'exceptions de Java.
- **Tracer** l'exécution d'un programme suite à l'exécution d'un énoncé throw.
- **Expliquer** l'affirmation suivante : les exceptions sont soit à déclaration obligatoire ou non obligatoire.
- **Modifier** une application afin qu'elle signifie les erreurs à l'aide d'exceptions.
- **Modifier** une application afin qu'elle gère les situations d'erreur.
- **Créer** de nouveaux types d'exceptions.

Lectures :

- Pages 29–36, 559, 608–619 de E. Koffman et P. Wolfgang.

Plan

Traitement des erreurs

Théorie : Traitement des erreurs

Objectifs d'apprentissage :

- **Distinguer** les erreurs de compilation des erreurs d'exécution
- **Développer** des préconditions pour une méthode de classe
- **Développer** des préconditions pour une méthode d'instance

Traitement des erreurs

Introduction

Désastres informatiques

- **Donnez** des exemples d'erreurs de programmation qui ont donnez lieu à des catastrophes.

Vehicules autonomes

Source : Grendelkhan

Erreurs de compilation et erreurs d'exécution

On distingue **deux** types d'erreurs : les erreurs de **compilation** et les erreurs d'**exécution**.

Compilation :

- Erreurs de **syntaxes**
- Java étant un langage **fortement typé**, le compilateur vérifie aussi le type de chaque expression, ce qui permet la détection de certaines erreurs le plus tôt possible, **avant l'exécution du programme**. La vérification des types permet de s'assurer que les opérations sur une valeur sont valides pour le type de la valeur.

Les **erreurs de compilation** n'affectent pas les usagers !

Discussion : Erreurs d'exécution

Donnez des exemples d'erreurs d'exécution !

Discussion : Sources d'erreurs d'exécution

Nommez les sources d'erreurs d'exécution !

*En conséquence, nous verrons que Java nous offre aussi deux façons de traiter les situations d'erreurs.

Traitement des erreurs

Préconditions

Précondition

- Une **précondition** est **condition préalable** à l'exécution d'une méthode.
- En programmation orientée objet, nous devons valider les **valeurs des paramètres**, mais aussi l'**état de l'objet**.

Une méthode **doit** débuter par la **validation** des **préconditions** !

Traitement des erreurs : Que faire ?

Caractéristiques recherchées

La détection et le traitement des situations d'erreurs contribuent à rendre les programmes plus **robustes**.

- **Indiquer** la source de l'erreur de façons précises.
  - Quelle méthode ? Quel énoncé ? Quelle est la nature de l'erreur ?
- **Forcer** le logiciel à prendre une action pour corriger la situation.
  (impossible d'ignorer les erreurs)

Exception

Exception

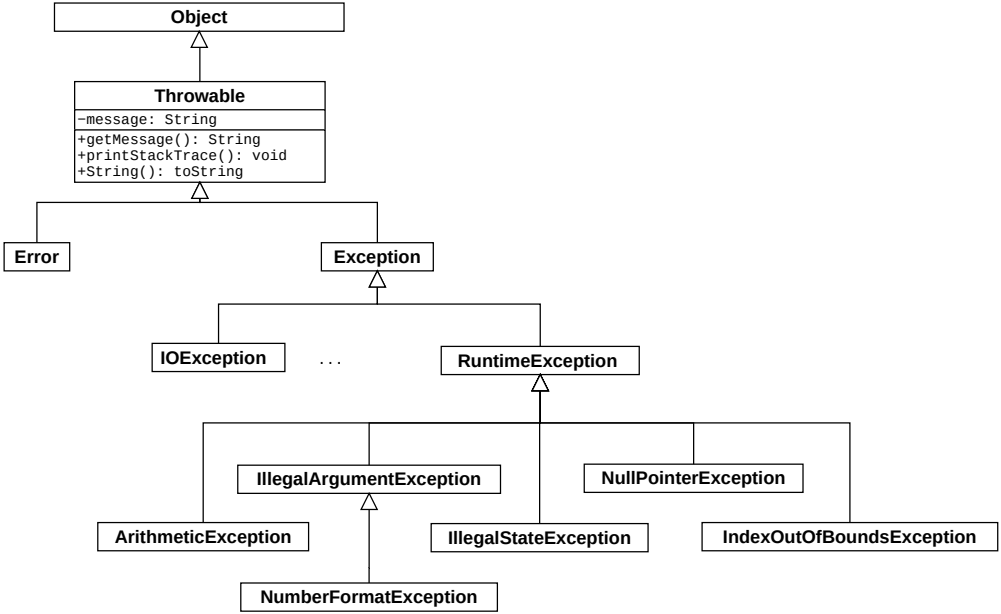
Objectifs d'apprentissage :

- **Résumer** le rôle de la classe **Exception**

Exception

La classe Exception

Exception est une classe !



Exception est une classe !

- Les exceptions sont des **objets !**
- Une situation d'erreur est modélisée à l'aide d'un objet de la classe **Throwable**, ou l'une de ses sous-classes.
  - L'objet encapsule un message d'erreur.
- Entre autres, la classe **Throwable** déclare les méthodes **String getMessage()** et **void printStackTrace()**.

Exception

Variable de type Exception

Déclarer une variable de type Exception

Déclarer une référence de type **Exception** n'a rien d'exceptionnel.

```
Exception e;
```

Exception

Objet de la classe Exception

Créer un objet de la classe Exception

De même, **créer** un objet de la classe **Exception** n'a rien d'exceptionnel.

```
e = new Exception("Houston, we've had a problem!");
```

La phrase « Houston, we've had a problem » est devenue célèbre à la suite du film « Apollo 13 ».

Throw

Signaler une situation d'erreur

Objectifs d'apprentissage :

- **Modifier** une application afin qu'elle signifie les erreurs à l'aide d'exceptions.
- **Tracer** l'exécution d'un programme suite à l'exécution d'un énoncé throw.

Lectures :

- Pages 559, 608–619 de E. Koffman et P. Wolfgang.

Signaler une situation d'erreur

- Considérons l'implémentation d'une pile à l'aide d'éléments chaînés et de sa méthode **pop()**.
- Le retrait d'un élément lorsque la pile est vide constitue une situation d'erreur.
- Formulez un test afin de valider la précondition de la méthode **pop**.

Throw

Énoncé throw

Énoncé throw

En Java, l'énoncé « **throw** » modifie le flot de contrôle normal. Son argument est une référence vers un objet de la classe **Throwable** ou l'une de ses sous-classes.

Énoncé throw

Throw

Transfert du contrôle

Transfert du contrôle

Lorsqu'une situation d'exception est signalée,

- L'énoncé ou l'expression **termine abruptement** ;
- Si l'exception n'est pas traitée, **la pile des appels de méthodes sera déroulée complètement**, c'est-à-dire que chaque appel de méthode se trouvant sur la pile d'exécution terminera abruptement, et l'exécution du programme se terminera avec l'impression de la pile des appels au moment de l'erreur («stack trace») ;
- **Aucun des énoncés** et **aucune des parties de l'expression** se trouvant après l'expression ayant causée l'erreur **ne seront exécutés** ;
- **Suite à une exception**, les prochains énoncés exécutés sont ceux se trouvant dans un bloc **catch** ou **finally**.

Throw

Exemples

```
class Test {
    public static void main( String[] args ) {
        System.out.println( "Label 1" );
        if (args.length == 0) {
            throw new RuntimeException("Houston...");
        }
        System.out.println( "Label 2" );
    }
}
```

Houston, we've had a problem !

- À la suite de l'énoncé **throw**, la méthode **termine** abruptement

```
> java Test
Label 1
Exception in thread "main" java.lang.RuntimeException:
Houston, we've had a problem!
        at Test.main(Test.java:5)
```

- Ainsi, la chaîne « Label 2 » **ne sera pas affichée** sur la console

```java
public class Test {
    public static boolean error(int v) {
        if (v == 0) {
            throw new RuntimeException("Oops! I'm sorry.");
        }
        return true;
    }
    public static boolean display() {
        System.out.println("Label 2");
        return true;
    }
    public static void main( String[] args ) {
        System.out.println("Label 1" );
        if (error(0) || display()) {
            System.out.println("Label 3");
        }
        System.out.println("Label 4");
    }
}
```

- Compilez et exécutez le programme ci-dessus.

Oops ! I'm sorry

- À la suite de l'énoncé **throw**, la méthode **error termine** abruptement, de même, la méthode **main termine** abruptement.

```
> java Test
Label 1
Exception in thread "main" java.lang.RuntimeException:
Oops! I'm sorry.
        at Test.error(Test.java:5)
        at Test.main(Test.java:16)
```

- Ainsi, les chaînes « Label 2 », « Label 3 » et « Label 4 » **ne seront pas affichées** sur la console

```java
public class Test {
    public static void c() {
        System.out.println("c: Label 1" );
        if (true) {
            throw new RuntimeException("dessus de la pile");
        }
        System.out.println("c: Label 2");
    }
    public static void b() {
        System.out.println("b: Label 1");
        c();
        System.out.println("b: Label 2");
    }
    public static void a() {
        System.out.println("a: Label 1");
        b();
        System.out.println("a: Label 2");
    }
    public static void main(String[] args) {
        System.out.println("main: Label 1");
        a();
        System.out.println("main: Label 2");
    }
}
```

Dérouler la pile des appels

```
> java Test
main: Label 1
a: Label 1
b: Label 1
c: Label 1
Exception in thread "main" java.lang.RuntimeException:
dessus de la pile des appels
        at Test.c(Test.java:6)
        at Test.b(Test.java:11)
        at Test.a(Test.java:16)
        at Test.main(Test.java:21)
```

```
public E pop() {
    if (isEmpty()) {
        throw new EmptyStackException();
    }
    E saved;
    saved = elems[--top];
    elems[top] == null;
    return saved;
}
```

- **Si** la **précondition** n'est pas satisfaite, la méthode signale une situation d'erreur.
- **Sinon**, la méthode **retire** le premier élément de la pile, et **retourne** sa valeur.

Try-Catch

Théorie

Objectifs d'apprentissage :

- **Modifier** une application afin qu'elle gère les situations d'erreur.
- **Tracer** l'exécution d'un programme suite à l'exécution d'un énoncé throw, mais en présence d'énoncés try-catch.

Lectures :

- Pages 559, 608–619 de E. Koffman et P. Wolfgang.

Try-Catch

Syntaxe

Traiter les exceptions

La construction syntaxique **try/catch** est utilisée afin de récupérer le contrôle lors de situations d'erreurs.

```
try {
    // ...
} catch (ExceptionType1 id1) {
    // statements;
} catch (ExceptionType2 id2) {
    // statements;
} finally {
    // statements;
}
```

Si aucune exception n'est lancée, seuls les énoncés du bloc **try** et du bloc **finally** seront exécutés.

Try-Catch

Exemple

```
public class Grill {
    private Burner burner = new Burner();
    public void cooking() {
        try {
            burner.on();
            addSteak();
            addSaltAndPepper();
            boolean done = false;
            while (! done) {
                done = checkSteak();
            }
        } catch (OutOfGazException e1) {
            callRetailer();
        } catch (FireException e2) {
            extinguishFire();
        } finally {
            burner.off();
        }
    }
}
```

Exemple : try/catch

```
int DEFAULT_VALUE = 0;
int value;

try {

    value = Integer.parseInt("100");

} catch (NumberFormatException e) {

    value = DEFAULT_VALUE;

}

System.out.println( "value = " + value );
```

Comment sait-on quel type d'exception sera lancé ?

parseInt

```
public static int parseInt(String s)
                    throws NumberFormatException
```

Parses the string argument as a signed decimal integer. The characters in the string must all be decimal digits, except that the first character may be an ASCII minus sign `'-'` (`'\u002D'`) to indicate a negative value or an ASCII plus sign `'+'` (`'\u002B'`) to indicate a positive value. The resulting integer value is returned, exactly as if the argument and the radix 10 were given as arguments to the `parseInt(java.lang.String, int)` method.

Parameters:

`s` - a String containing the int representation to be parsed

Returns:

the integer value represented by the argument in decimal.

Throws:

`NumberFormatException` - if the string does not contain a parsable integer.

Exemple : try/catch

```
int DEFAULT_VALUE = 0;
int value;

try {

    value = Integer.parseInt("cent");

} catch (NumberFormatException e) {

    value = DEFAULT_VALUE;

}

System.out.println( "value = " + value );
```

Try-Catch

Flot de contrôle

Flot de contrôle

- Lorsqu'une exception est lancée, l'exécution des énoncés du bloc **try** se termine (abruptement) et se poursuit avec les énoncés du premier bloc **catch** dont le paramètre est du même type que celui de l'objet modélisant la situation d'erreur, ou d'un type plus général, suivi de l'exécution des énoncés du bloc **finally**, si présent.
- Aucun autre bloc ne sera exécuté.
- Si aucun bloc **catch** n'est adéquat alors l'exception percole.
- Les énoncés du bloc **finally** sont toujours exécutés : avec ou sans erreur.
  - Les blocs finally sont utilisés afin fermer des fichiers ouverts, par exemple, ou de façon générale, afin de traiter les «post»-conditions.

Try-Catch

Exemple complet

```java
public class Test {
    public static void c() {
        System.out.println( "c() :: about to throw exception" );
        throw new RuntimeException( "from c()" );
    }
    public static void b() {
        System.out.println( "b() :: pre-" );
        c();
        System.out.println( "b() :: post-" );
    }
    public static void a() {
        System.out.println( "a() :: pre-" );
        try {
            b();
        } catch ( RuntimeException e ) {
            System.out.println( "a() :: caught exception" );
        }
        System.out.println( "a() :: calling b, no try block" );
        b();
        System.out.println( "a() :: post-" );
    }
    public static void main( String[] args ) {
        System.out.println( "main( ... ) :: pre-" );
        a();
        System.out.println( "main( ... ) :: post-" );
    }
}
```

Try-Catch

La référence de type Exception

Au sujet du paramètre du bloc catch

```java
int DEFAULT_VALUE = 0;
int value;

try {
    value = Integer.parseInt("douze");
} catch (NumberFormatException e) {
    System.out.println("warning: " + e.getMessage());
    value = DEFAULT_VALUE;
}
```

- Le paramètre **e** est une référence désignant l'objet passé à l'énoncé **throw**, comme pour tout autre objet, la notation pointée est utilisée afin d'accéder aux méthodes de l'objet.

Throws

Théorie

Objectifs d'apprentissage :

- **Expliquer** l'affirmation suivante : les exceptions sont soit à déclaration obligatoire ou non obligatoire.

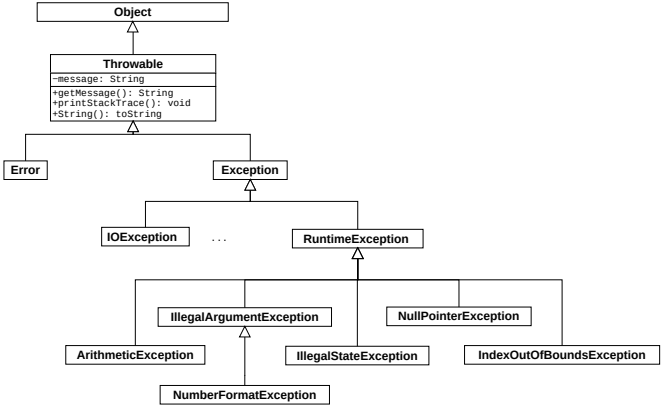
Lectures :

- Pages 559, 608–619 de E. Koffman et P. Wolfgang.

Throws

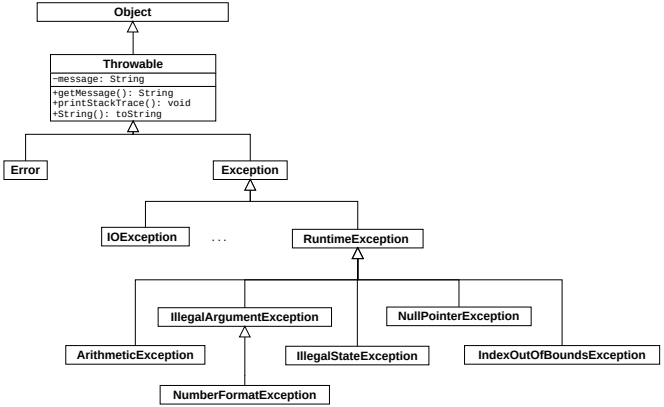
Déclaration obligatoire ou non

Déclaration obligatoire ou déclaration non obligatoire



- Les descendants de la classe **Exception** sont à **déclaration obligatoire**, on dit *«checked»* en anglais.

Déclaration obligatoire ou déclaration non obligatoire



- **Sauf**, si elles sont des sous-classes de la classe **RuntimeExcetion**, elle sont alors à **déclaration non obligatoire**. En anglais, on dit *«unchecked»*.

Discussion : Sources d'erreurs d'exécution

Exceptions à déclaration obligatoire

Une méthode faisant appel à une méthode pouvant lancer une exception à **déclaration obligatoire** doit :

- Traiter l'exception (**catch**), ou ;
- Laisser passer l'exception et la déclarer (**throws**).

Throws

Syntaxe

Déclarer une exception

- On utilise l'énoncé **throws** pour déclarer une ou plusieurs exceptions.
  - Ici, nous informons les utilisateurs de la méthode **do** qu'elle pourrait lancer une exception de type **IOException**.

```
public static void do(String name) throws IOException {

    // ...
}
```

Throws

Exemple

```java
import java.io.*;

public class Keyboard {
    public static int getInt() {
        byte[] buffer = new byte[ 256 ];

        System.in.read(buffer); // throws IOException

        String s = new String(buffer);
        int num = Integer.parseInt(s.trim());
        return num;
    }
    public static void main(String[] args) {
        System.out.print("Please enter a number: ");
        int n = Keyboard.getInt();
        System.out.println("Number is: " + n);
    }
}
```

Erreur de compilation

- Les exceptions à **déclaration obligatoire** doivent être **traitées** ou **déclarées**.
  - Sinon, elles causent des erreurs de compilation.

```
> javac Keybord.java
Keyboard.java:9: unreported exception java.io.IOException;
must be caught or declared to be thrown
     System.in.read(buffer);
                    ^
1 error
```

```java
import java.io.*;

public class Keyboard {
    public static int getInt() throws IOException {
        byte[] buffer = new byte[256];

        System.in.read(buffer); // throws IOException

        String s = new String(buffer);
        int num = Integer.parseInt(s.trim());
        return num;
    }
    public static void main(String[] args) {
        System.out.print("Please enter a number: ");

        int n = Keyboard.getInt(); // throws IOException

        System.out.println("Number is: " + n);
    }
}
```

Erreur de compilation

- Les exceptions à **déclaration obligatoire** doivent être **traitées** ou **déclarées**.
  - Sinon, elles causent des erreurs de compilation.

```
> javac Keyboard.java
Keyboard.java:22: unreported java.io.IOException;
must be caught or declared to be thrown
     int n = Keyboard.getInt();
                              ^
1 error
```

```java
import java.io.*;

public class Keyboard {
    public static int getInt() throws IOException {
        byte[] buffer = new byte[256];

        System.in.read(buffer); // throws IOException

        String s = new String(buffer);
        int num = Integer.parseInt(s.trim());
        return num;
    }
    public static void main(String[] args)
    throws IOException {

        System.out.print("Please enter a number: ");

        int n = Keyboard.getInt(); // throws IOException

        System.out.println("Number is: " + n );
    }
}
```

Déclarer une exception

- Lorsqu'on **déclare** (**throws**) une exception sans la traitée (**catch**) la méthode termine abruptement lorsque l'exception est lancée.

```
> java Keyboard
Please enter a number: oups
Exception in thread "main" java.lang.NumberFormatException
For input string: "oups"
     at java.lang.NumberFormatException.
        forInputString(NumberFormatException.java:48)
     at java.lang.Integer.parseInt(Integer.java:468)
     at java.lang.Integer.parseInt(Integer.java:518)
     at Keyboard.getInt(Keyboard.java:13)
     at Keyboard.main(Keyboard.java:23)
```

```java
import java.io.*;

public class Keyboard {

    public static int getInt() throws IOException {
        byte[] buffer = new byte[256];

        System.in.read(buffer);

        String s = new String(buffer);

        int num = Integer.parseInt(s.trim());

        return num;
    }

    // ...
```

```
    // ...

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

        int n;
        boolean done = false;

        while (! done) {
            System.out.print("Please enter a number: ");
            try {
                n = Keyboard.getInt();
                System.out.println("The number is " + n);
                done = true;
            } catch (NumberFormatException e) {
                System.out.println("Not a number!");
            }
        }
    }
}
```

Exemple

- Cet exemple traite les exceptions de type **NumberFormatException**, mais laisse passer les exceptions de type **IOException**.

```
> java Keyboard
Please enter a number: oups
Not a number!
Please enter a number: a1
Not a number!
Please enter a number:  1
The number is 1
```

Nouveaux types

Théorie

Objectifs d'apprentissage :

- **Créer** de nouveaux types d'exceptions.

Lectures :

- Pages 29–36 de E. Koffman et P. Wolfgang.

Nouveaux types

Syntaxe

Comment crée-t-on de nouveaux types d'exception ?

- Les exceptions sont des **objets**.
- Il suffit donc de **créer de nouvelles classes**.
- Les sous-classes de la classe **Exception** sont à **déclaration obligatoire**.
- Sauf, si elles sont des sous-classes de la classe **RuntimeException**, alors elles sont à **déclaration non obligatoire**.

Créer de nouveaux types d'exception

```java
public class MyException extends Exception {
}
```

```java
public class MyException extends RuntimeException {
    public MyException() {
        super();
    }
    public MyException(String message) {
        super(message);
    }
}
```

- **Pourquoi** créer de nouveaux types d'exception ?

Nouveaux types

Exemple

Exemple : Time

- Dans l'exemple qui suit, la méthode **parseTime** attrape les exceptions de type **NumberFormatException** ou **NoSuchElementException** afin de lancer une exception dont le type est plus informatif, **TimeFormatException**.

```
public class TimeFormatException extends IllegalArgumentException {

    public TimeFormatException() {
        super();
    }

    public TimeFormatException(String msg) {
        super(msg);
    }
}
```

Exemple : Time

```
public class Time {

    // ...

    public static Time parseTime( String timeString ) {

        StringTokenizer st;
        st = new StringTokenizer(timeString, ":", true);

        int h, m, s;

        try {
            h = Integer.parseInt(st.nextToken ());
        } catch (NumberFormatException e1) {
            throw new TimeFormatException("not a number: "+timeString);
        } catch (NoSuchElementException e2) {
            throw new TimeFormatException("separator not found: "+timeString);
        }

        try {
            st.nextToken();
        } catch (NoSuchElementException e2) {
            throw new TimeFormatException("separator not found: "+timeString);
        }
```

Exemple : Time

```
        try {
            m = Integer.parseInt(st.nextToken());
        } catch (NumberFormatException e1) {
            throw new TimeFormatException("not a number: "+timeString);
        } catch (NoSuchElementException e2) {
            throw new TimeFormatException("separator not found: "+timeString);
        }

        try {
            st.nextToken();
        } catch (NoSuchElementException e2) {
            throw new TimeFormatException("separator not found: "+timeString);
        }

        try {
            s = Integer.parseInt(st.nextToken());
        } catch (NumberFormatException e1) {
            throw new TimeFormatException("not a number: "+timeString);
        } catch (NoSuchElementException e2) {
            throw new TimeFormatException("third field not found: "+timeString);
        }
```

Exemple : Time

```
        if (st.hasMoreTokens()) {
            throw new TimeFormatException("invalid suffix:" + timeString);
        }

        if ((h<0) || (h>23) || (m<0) || (m>59) || (s<0) || (s>59) ) {
            throw new TimeFormatException( "values out of range:" + timeString);
        }

        return new Time(h,m,s);
    }
}
```

Résumé

- Une **précondition** est condition **préalable** à l'exécution d'une méthode.
  - En programmation orientée objet, nous devons valider les **valeurs des paramètres**, mais aussi l'**état de l'objet**.
- En Java, on utilise les **exceptions** afin de signaler une **erreur d'exécution**
- Le bloc **try/catch** est utilisé pour traiter les exceptions, c.-à-d. stopper la propagation.

Prochain module

- Type abstrait de données (TAD) : **file**.

References I


E. B. Koffman and Wolfgang P. A. T.
Data Structures : Abstraction and Design Using Java.
John Wiley & Sons, 3e edition, 2016.

Marcel **Turcotte**

Marcel.Turcotte@uOttawa.ca

École de **science informatique** et de génie électrique (SIGE)

Université d'Ottawa