String, réflexivité et Exceptions

Victor Marsault
Aldric Degorre

CPOO 2015

Outline

1 Manipulation de String

2 Un tout petit peu de réflexivité

3 L'exception est la norme

Séparation de chaîne

- Permet de séparer une chaîne en des sous-chaînes suivant un certain séparateur.
- Outil d'analyse syntaxique (parsing en anglais) basique mais déjà très puissant.
- Utilisation pour
  - Fichiers de configurations.
  - Passage de message (bas-niveau) par le réseau.
  - Bien d'autres choses.

```
String {
  ...
    String[] split(String separateur)
    String[] split(String separateur, int limite)
  ...
}
```

Exemples


```
static public void  main(String args) {
    String str = "foo:bar bar:foo.foo";
```

Exemples


```
static public void  main(String args) {
    String str = "foo:bar bar:foo.foo";

    String[] tab1 = str.split(":");
    //["foo","bar bar","foo.foo"]
```

Exemples


```
static public void  main(String args) {
    String str = "foo:bar bar:foo.foo";

    String[] tab1 = str.split(":");
    //["foo","bar bar","foo.foo"]

    String[] tab2 = str.split("bar");
    //["foo:", " ", ":foo.foo"]
```

Exemples


```
static public void  main(String args) {
    String str = "foo:bar bar:foo.foo";

    String[] tab1 = str.split(":");
    //["foo","bar bar","foo.foo"]

    String[] tab2 = str.split("bar");
    //["foo:", " ", ":foo.foo"]

    String[] tab3 = str.split("o");
    //["f", "", ":bar bar:f", "", ".f"]
    // split enleve les "" de fin
```

Exemples


```
static public void  main(String args) {
    String str = "foo:bar bar:foo.foo";

    String[] tab1 = str.split(":");
    //["foo","bar bar","foo.foo"]

    String[] tab2 = str.split("bar");
    //["foo:", " ", ":foo.foo"]

    String[] tab3 = str.split("o");
    //["f", "", ":bar bar:f", "", ".f"]
    // split enleve les "" de fin

    String[] tab4 = str.split("."); //[]
    //L'argument est considéré comme une expression régu-
    //-lière donc '.' veut dire n'importe quelle lettre.
    //Les caractères ^.[]*$()?|+{} sont spéciaux
```

Exemples


```
static public void  main(String args) {
    String str = "foo:bar bar:foo.foo";

    String[] tab1 = str.split(":");
    //["foo","bar bar","foo.foo"]

    String[] tab2 = str.split("bar");
    //["foo:", " ", ":foo.foo"]

    String[] tab3 = str.split("o");
    //["f", "", ":bar bar:f", "", ".f"]
    // split enleve les "" de fin

    String[] tab4 = str.split("."); //[]
    //L'argument est considéré comme une expression régu-
    //-lière donc '.' veut dire n'importe quelle lettre.
    //Les caractères ^.[]*$()?|+{} sont spéciaux

    String[] tab4 = str.split("[.]");
    //["foo:bar bar:foo","foo"]
}
```

Exemples (2)


```
static public void  main(String args) {
    String str = "foo:bar bar:foo.foo";
```

Exemples (2)


```
static public void  main(String args) {
    String str = "foo:bar bar:foo.foo";

    String[] tab1 = str.split(":",2); //2 résultats max
    //["foo", "bar bar:foo.foo"]
```

Exemples (2)


```
static public void  main(String args) {
    String str = "foo:bar bar:foo.foo";

    String[] tab1 = str.split(":",2); //2 résultats max
    //["foo", "bar bar:foo.foo"]

    String[] tab2 = str.split("o",6);
    //["f", "", ":bar bar:f", "", ".f", "o"]
```

Exemples (2)


```
static public void  main(String args) {
    String str = "foo:bar bar:foo.foo";

    String[] tab1 = str.split(":",2); //2 résultats max
    //["foo", "bar bar:foo.foo"]

    String[] tab2 = str.split("o",6);
    //["f", "", ":bar bar:f", "", ".f", "o"]

    String[] tab3 = str.split("o",7); //ou >7
    //["f", "", ":bar bar:f", "", ".f", "", ""]
```

Exemples (2)


```
static public void  main(String args) {
    String str = "foo:bar bar:foo.foo";

    String[] tab1 = str.split(":",2); //2 résultats max
    //["foo", "bar bar:foo.foo"]

    String[] tab2 = str.split("o",6);
    //["f", "", ":bar bar:f", "", ".f", "o"]

    String[] tab3 = str.split("o",7); //ou >7
    //["f", "", ":bar bar:f", "", ".f", "", ""]

    String[] tab4 = str.split("o",0); //comme str.split("o")
    //["f", "", ":bar bar:f", "", ".f"]
```

Exemples (2)


```
static public void  main(String args) {
    String str = "foo:bar bar:foo.foo";

    String[] tab1 = str.split(":",2); //2 résultats max
    //["foo", "bar bar:foo.foo"]

    String[] tab2 = str.split("o",6);
    //["f", "", ":bar bar:f", "", ".f", "o"]

    String[] tab3 = str.split("o",7); //ou >7
    //["f", "", ":bar bar:f", "", ".f", "", ""]

    String[] tab4 = str.split("o",0); //comme str.split("o")
    //["f", "", ":bar bar:f", "", ".f"]

    String[] tab5 = str.split("o",-1);
    //["f", "", ":bar bar:f", "", ".f", "", ""]
    // si négatif, illimité et laisse les "" à la fin.
```

Exemples (2)


```
static public void  main(String args) {
    String str = "foo:bar bar:foo.foo";

    String[] tab1 = str.split(":",2); //2 résultats max
    //["foo", "bar bar:foo.foo"]

    String[] tab2 = str.split("o",6);
    //["f", "", ":bar bar:f", "", ".f", "o"]

    String[] tab3 = str.split("o",7); //ou >7
    //["f", "", ":bar bar:f", "", ".f", "", ""]

    String[] tab4 = str.split("o",0); //comme str.split("o")
    //["f", "", ":bar bar:f", "", ".f"]

    String[] tab5 = str.split("o",-1);
    //["f", "", ":bar bar:f", "", ".f", "", ""]
    // si négatif, illimité et laisse les "" à la fin.
}
```

Un mot sur le remplacement


```
class String {
    String replace(char ancienChar, char nouveauChar);
    public String replace(CharSequence ancienneChaine,
                          CharSequence nouvelleChaine);
    //CharSequence est une interface implémentée par String
    // "aaab".replace("aa","b") renvoie "bab".
    String replaceAll(String regex, String nouvelleChaine);
    String replaceFirst(String regex, String nouvelleChaine);
}
```

Formattage de Chaine

- Hérité de C (printf, sprintf, etc.)
- Sert à afficher une chaîne formatée (comprenant d'autres données à l'intérieur)
- Donne une vision générale du résultat, épuré des différentes variables. (Contrairement à, par exemple, `"Le repertoire contenant "+f.getPath()+"est"+f.getParent()+"."`)

```
public static String format(String format, Object... args);
```

String.format par l'exemple


```
public static void main(String[] args) {
    String motif =
          "La chaine \"%s\" est utilisée dans %f%%%n"
          + "des exemples; le caractere '%c' est %d"
          + " fois plus%nutilisé que '%c'.";
    String resultat = String.format(motif, "Hello Word",
                                    42.83, 'e', 18, 'z');
```

String.format par l'exemple


```
public static void main(String[] args) {
    String motif =
          "La chaine \"%s\" est utilisée dans %f%%%n"
          + "des exemples; le caractere '%c' est %d"
          + " fois plus%nutilisé que '%c'.";
    String resultat = String.format(motif, "Hello Word",
                                    42.83, 'e', 18, 'z');
    System.out.println(resultat);
}
```

```
La chaine "Hello Word" est utilisée dans 42,830000%
des exemples; le caractere 'e' est 18 fois plus
utilisé que 'z'
```

String.format par l'exemple


```
public static void main(String[] args) {
    String motif =
          "La chaine \"%s\" est utilisée dans %f%%%n"
          + "des exemples; le caractere '%c' est %d"
          + " fois plus%nutilisé que '%c'.";
    String resultat = String.format(motif, "Hello Word",
                                    42.83, 'e', 18, 'z');
    System.out.println(resultat);
}
```

```
La chaine "Hello Word" est utilisée dans 42,830000%
des exemples; le caractere 'e' est 18 fois plus
utilisé que 'z'
```

%s	attend String	%d	attend int (décimal)
%f	attend double (float)	%c	attend char
%b	attend bool		
%n	affiche une fin de ligne	%%	affiche %

Outline

1 Manipulation de String

2 Un tout petit peu de réflexivité

3 L'exception est la norme

Le type `Class`

- est une sous-classe d'`Object` ;
- contient tous les entêtes de la classe (constructeurs, méthodes, sous-classes, etc.) ;
- permet de créer des nouvelles instances de la classe, d'appeler des méthodes à partir de leurs noms.

Quels services fournit `Class<T>`


```
static Class forName(String nom); //String -> Class
String getName(); //Class -> String
```

Quels services fournit `Class<T>`


```
static Class forName(String nom); //String -> Class
String getName(); //Class -> String

Class<?>[] getClasses();//classes internes
Constructor<?>[] getConstructors(); //constructeurs publiques
Class<?>[] getInterfaces(); //interfaces implementees
Method[] getMethods(); //méthodes publiques
Field[] getFiels(); //attributs publiques
```

Quels services fournit `Class<T>`


```
static Class forName(String nom); //String -> Class
String getName(); //Class -> String

Class<?>[] getClasses();//classes internes
Constructor<?>[] getConstructors(); //constructeurs publiques
Class<?>[] getInterfaces(); //interfaces implementees
Method[] getMethods(); //méthodes publiques
Field[] getFiels(); //attributs publiques

Method getMethod(String name, Class<?>... parameterTypes);
Object invoke(Object obj, Object... args); //dans Method
```

Quels services fournit Class<T>


```
static Class forName(String nom); //String -> Class
String getName(); //Class -> String

Class<?>[] getClasses();//classes internes
Constructor<?>[] getConstructors(); //constructeurs publiques
Class<?>[] getInterfaces(); //interfaces implementees
Method[] getMethods(); //méthodes publiques
Field[] getFiels(); //attributs publiques

Method getMethod(String name, Class<?>... parameterTypes);
Object invoke(Object obj, Object... args); //dans Method

Constructor<T> getConstructor(Class<?>... parameterTypes);
T newInstance(Object... initargs);//dans Constructor
```

Quels services fournit `Class<T>`


```
static Class forName(String nom); //String -> Class
String getName(); //Class -> String

Class<?>[] getClasses();//classes internes
Constructor<?>[] getConstructors(); //constructeurs publiques
Class<?>[] getInterfaces(); //interfaces implementees
Method[] getMethods(); //méthodes publiques
Field[] getFiels(); //attributs publiques

Method getMethod(String name, Class<?>... parameterTypes);
Object invoke(Object obj, Object... args); //dans Method

Constructor<T> getConstructor(Class<?>... parameterTypes);
T newInstance(Object... initargs);//dans Constructor
```

Toutes ces fonctions lèvent beaucoup d'exceptions, à utiliser avec modération.

Pseudo-interface

Le but est d'écrire une méthode qui lance `execute(String str, int i)` si l'Objet donné en argument admet une telle méthode.

Pseudo-interface

Le but est d'écrire une méthode qui lance `execute(String str, int i)` si l'Objet donné en argument admet une telle méthode.

Il ne faut jamais faire une chose pareille.

Pseudo-interface

Le but est d'écrire une méthode qui lance `execute(String str, int i)` si l'Objet donné en argument admet une telle méthode.

Il ne faut jamais faire une chose pareille.

```
public static void executeObj(Object o, String str, int i)
throws Exception {
    Class c = o.getClass();
    //méthode finale hérité d'Object
    //renvoie la classe a l'execution
```

Pseudo-interface

Le but est d'écrire une méthode qui lance `execute(String str, int i)` si l'Objet donné en argument admet une telle méthode.

Il ne faut jamais faire une chose pareille.

```
public static void executeObj(Object o, String str, int i)
throws Exception {
    Class c = o.getClass();
    //méthode finale hérité d'Object
    //renvoie la classe a l'execution
```

Pseudo-interface

Le but est d'écrire une méthode qui lance `execute(String str, int i)` si l'Objet donné en argument admet une telle méthode.

Il ne faut jamais faire une chose pareille.

```
public static void executeObj(Object o, String str, int i)
throws Exception {
    Class c = o.getClass();
    //méthode finale hérité d'Object
    //renvoie la classe a l'execution

    Method m =
        c.getMethod("execute", String.class, int.class);
    // .class donne la classe depuis un type
```

Pseudo-interface

Le but est d'écrire une méthode qui lance `execute(String str, int i)` si l'Objet donné en argument admet une telle méthode.

Il ne faut jamais faire une chose pareille.

```
public static void executeObj(Object o, String str, int i)
throws Exception {
    Class c = o.getClass();
    //méthode finale hérité d'Object
    //renvoie la classe a l'execution

    Method m =
        c.getMethod("execute", String.class, int.class);
    // .class donne la classe depuis un type
    m.invoke(o, str, i); // appel de o.m(str,i)
}
```

Pseudo-interface (2)


```
class Main {
    static class Exec {
        public void execute(String str, int i) {
            System.out.println(str);}
    }
```

Pseudo-interface (2)


```
class Main {
    static class Exec {
        public void execute(String str, int i) {
            System.out.println(str);}
    }
    static class ExecDeux {
        public void execute(String str, int i) {
            System.out.println(i);}
    }
```

Pseudo-interface (2)


```
class Main {
    static class Exec {
        public void execute(String str, int i) {
            System.out.println(str);}
    }
    static class ExecDeux {
        public void execute(String str, int i) {
            System.out.println(i);}
    }
    public static void main(String args) {
        String str= "str";   Exec e= new Exec();
        ExecDeux e2= new ExecDeux();   int i= 0;
```

Pseudo-interface (2)


```
class Main {
    static class Exec {
        public void execute(String str, int i) {
            System.out.println(str);}
    }
    static class ExecDeux {
        public void execute(String str, int i) {
            System.out.println(i);}
    }
    public static void main(String args) {
        String str= "str";   Exec e= new Exec();
        ExecDeux e2= new ExecDeux();   int i= 0;
        executeObject(e, str, i); //affiche "str"
```

Pseudo-interface (2)


```
class Main {
    static class Exec {
        public void execute(String str, int i) {
            System.out.println(str);}
    }
    static class ExecDeux {
        public void execute(String str, int i) {
            System.out.println(i);}
    }
    public static void main(String args) {
        String str= "str";   Exec e= new Exec();
        ExecDeux e2= new ExecDeux();   int i= 0;
        executeObject(e, str, i); //affiche "str"
        executeObject(e2, str, i); //affiche "0"
```

Pseudo-interface (2)


```
class Main {
    static class Exec {
        public void execute(String str, int i) {
            System.out.println(str);}
    }
    static class ExecDeux {
        public void execute(String str, int i) {
            System.out.println(i);}
    }
    public static void main(String args) {
        String str= "str";   Exec e= new Exec();
        ExecDeux e2= new ExecDeux();   int i= 0;
        executeObject(e, str, i); //affiche "str"
        executeObject(e2, str, i); //affiche "0"
        executeObject(str, str, i); //plante salement
    }
}
```

Cas particulier des enum


```
public enum Direction {
    Nord, Sud, Est, Ouest;
}

public static void main (String[] args) {
    Direction sud = Direction.valueOf("Sud");
    //String->instance (d'enum)
```

Cas particulier des enum


```
public enum Direction {
    Nord, Sud, Est, Ouest;
}

public static void main (String[] args) {
    Direction sud = Direction.valueOf("Sud");
    //String->instance (d'enum)

    int i = sud.getOrdinal() //1
    //instance->int
```

Cas particulier des enum


```
public enum Direction {
    Nord, Sud, Est, Ouest;
}

public static void main (String[] args) {
    Direction sud = Direction.valueOf("Sud");
    //String->instance (d'enum)

    int i = sud.getOrdinal() //1
    //instance->int

    Direction est = Direction.values()[2] //Est
    //enum-> instance[]  donc  int -> instance
```

Cas particulier des enum


```
public enum Direction {
    Nord, Sud, Est, Ouest;
}

public static void main (String[] args) {
    Direction sud = Direction.valueOf("Sud");
    //String->instance (d'enum)

    int i = sud.getOrdinal() //1
    //instance->int

    Direction est = Direction.values()[2] //Est
    //enum-> instance[]  donc  int -> instance
    //Deconseillé par la documention java
```

Cas particulier des enum


```
public enum Direction {
    Nord, Sud, Est, Ouest;
}

public static void main (String[] args) {
    Direction sud = Direction.valueOf("Sud");
    //String->instance (d'enum)

    int i = sud.getOrdinal() //1
    //instance->int

    Direction est = Direction.values()[2] //Est
    //enum-> instance[]  donc  int -> instance
    //Deconseillé par la documention java

    String str = est.name(); //"Est"
    // instance->String
}
```

Outline

1 Manipulation de String

2 Un tout petit peu de réflexivité

3 L'exception est la norme

Quand on n'a pas d'exceptions

- En cas d'erreurs graves :
  - soit il se passe n'importe quoi ;
    (exemple : dépassement d'indice dans les tableaux : utilise la valeur d'une autre variable.)
  - soit : segmentation fault.

Quand on n'a pas d'exceptions

- En cas d'erreurs graves :
  - soit il se passe n'importe quoi ;
    (exemple : dépassement d'indice dans les tableaux : utilise la valeur d'une autre variable.)
  - soit : segmentation fault.
- Difficulté pour débugguer.

Quand on n'a pas d'exceptions

- En cas d'erreurs graves :
  - soit il se passe n'importe quoi ;
    (exemple : dépassement d'indice dans les tableaux : utilise la valeur d'une autre variable.)
  - soit : segmentation fault.
- Difficulté pour débugguer.
- Emploi de solutions impropre : renvoyer -1 ou un pointeur nul pour indiquer une erreur.

Exception (2)

Avantages

- Permet de mieux comprendre ce qu'il s'est passé.
- Permet de traiter un problème attendu, *au bon endroit*.
- Permet de "sauter en dehors de plusieurs boucles".

Inconvénients

- Syntaxe lourde (surtout en java).
- L'abus d'exceptions est mauvais pour la compréhension (effet "plat de nouilles").

Exception (2)

Avantages

- Permet de mieux comprendre ce qu'il s'est passé.
- Permet de traiter un problème attendu, *au bon endroit*.
- Permet de "sauter en dehors de plusieurs boucles".

Inconvénients

- Syntaxe lourde (surtout en java).
- L'abus d'exceptions est mauvais pour la compréhension (effet "plat de nouilles").

- `NullPointerException` $\sim$ segmentation fault.

Capturer une exception : try/catch


```
public void maMethode()  {
    try {
        // bloc de code pouvant lever une exception
    } catch MonException e {
        // traiter (ou non) l'exception.
    }
```

Capturer une exception : try/catch


```
public void maMethode()  {
    try {
        // bloc de code pouvant lever une exception
    } catch MonException e {
        // traiter (ou non) l'exception.
    } catch MonAutreException e {
        e.printStackTrace(); //affiche le message usuel
        exit(1); // quitte le programme avec une erreur
    }
```

Capturer une exception : try/catch


```
public void maMethode()  {
    try {
        // bloc de code pouvant lever une exception
    } catch MonException e {
        // traiter (ou non) l'exception.
    } catch MonAutreException e {
        e.printStackTrace(); //affiche le message usuel
        exit(1); // quitte le programme avec une erreur
    } finally {
        // bloc de code s'exécutant quoi qu'il arrive:
        // qu'une exception soit levée, capturée ou non.

        // sert typiquement à fermer les fichiers,
        // détruire les fichiers temporaires, etc.
    }
}
```

Capturer une exception : try/catch


```
public void maMethode() throws MaTroisiemeException {
    try {
        // bloc de code pouvant lever une exception
    } catch MonException e {
        // traiter (ou non) l'exception.
    } catch MonAutreException e {
        e.printStackTrace(); //affiche le message usuel
        exit(1); // quitte le programme avec une erreur
    } finally {
        // bloc de code s'exécutant quoi qu'il arrive:
        // qu'une exception soit levée, capturée ou non.

        // sert typiquement à fermer les fichiers,
        // détruire les fichiers temporaires, etc.
    }
}
```

`RuntimeException extends Exception`

- Exceptions qui auraient pu être évitées par l'utilisateur.
- Pas besoin d'être déclarée avec `throws`.
- Ne devrait pas se produire : "Exception Fatale".

Ex : `NullPointerException`, `IndexOutOfBoundsException`

`Exception` n'héritant pas de `RuntimeException`

- Exceptions censée "être capturée".
- Déclaration nécessaire avec `throws`.

Ex : `IOException`.

`RuntimeException extends Exception`

- Exceptions qui auraient pu être évitées par l'utilisateur.
- Pas besoin d'être déclarée avec `throws`.
- Ne devrait pas se produire : "Exception Fatale".

Ex : `NullPointerException`, `IndexOutOfBoundsException`

`Exception` n'héritant pas de `RuntimeException`

- Exceptions censée "être capturée".
- Déclaration nécessaire avec `throws`.

Ex : `IOException`.

`Error`

- Problème fatal généralement ingérable dans le code.
- Pas besoin d'être déclarée avec `throws`.

Ex : `StackOverflowError`.

Comment rendre ce code "sûr"


```
public static void main (String[] args) {
  String str="une chaine";
  System.out.println("La chaine courante est: "+str);
  System.out.print("Quel caractere voulez vous? ");
  Scanner sc = new Scanner(System.in);
  int i = sc.nextInt();
  // Peut lever InputMismatchException,
  // NoSuchElementException et IllegalStateException
  // (qui sont toutes runtime).
  System.out.println();
  System.out.println("Le caractere est: "+str.charAt(i));
  // Peut lever IndexOutOfBoundsException.
}
```

Solution sans exception


```
public static void main (String args) {
    String str= "une chaine";
    System.out.println("La chaine courante est: "+str);
    Scanner sc = new Scanner(System.in);
    while (true) {
        System.out.print("Quel caractere voulez vous? ");
        if (sc.hasNextInt()) {
            int i = sc.nextInt();
            if ((0<i) && (i<str.length())){
              System.out.println("\nLe caractere est: "
                    +str.charAt(i));
              break;
            }
        } else
            sc.next();
        System.out.print("Mauvaise entrée. ");
    }
}
```

Solution avec exception


```
static public void main (String args) {
    String str= "une chaine";
    System.out.println("La chaine courante est: "+str);
    Scanner sc = new Scanner(System.in);

    while (true) {
        try {
            System.out.print("Quel caractere voulez vous? ");
            int i =  sc.nextInt();
            System.out.println("\nLe caractere est: "
                  +str.charAt(i));
            break;
        } catch (InputMismatchException e) {
            sc.next();
        } catch (IndexOutOfBoundsException e) {     }
        System.out.print("Mauvaise entrée. ");
    }
}
```

Déclarer ses propres exceptions


```
public class monException extends Exception {
    public monException(String msg) {
        super(msg);
    }
    //Il n'y a usuellement rien d'autre.
}
```

Déclarer ses propres exceptions


```
public class monException extends Exception {
    public monException(String msg) {
        super(msg);
    }
    //Il n'y a usuellement rien d'autre.
}
```

```
public class monException extends RuntimeException {
    public monException(String msg) {
        super(msg);
    }
    //Il n'y a usuellement rien d'autre.
}
```