Chapitre VII – Documenter un projet

Chapitre VII – Documenter un projet

Généralités

On peut annoter son programme pour obtenir un fichier html de commentaires.

- commencer par $/**$, terminer par $*/$ (entre les deux commencer chaque ligne par $*$).
- @auteur : il peut y avoir plusieurs auteurs
- @see : pour créer un lien vers un autre document
- @param : pour indiquer les paramètres d'une méthode
- @exception :pour indiquer quelle exception est levée
- @return : pour indiquer la valeur de retour d'une méthode
- @version : pour donner le numéro de la version du code
- @since : pour donner le numéro de la version initiale
- @deprecated : pour indiquer qu'une méthode ne devrait plus être utilisée, (cela crée un warning à la compilation)

commande

Après compilation de `MonProgramme.java`, la commande
`javadoc MonProgramme.java`
engendre un fichier `MonProgramme.html`.
Options de la commande `javadoc` :

- -author : Indique que les commentaires tagés par `@author` devront être utilisées pour générer la documentation (par défaut, ces informations ne sont pas utilisées).
- -d repertoire : Permet de préciser le repertoire où javadoc placera les fichiers HTML générés (par défaut, répertoire courant).
- -public : Ne documente que les membres (méthodes, constructeurs, attributs) publics.
- -private : Documente tous les membres (méthodes, constructeurs, attributs), quelle que soit leur visibilité (par défaut membres publics et protected)

Un exemple

```
import java.util.*;
/** Le premier exemple de programme  Java.
* Affiche une chaîne de caractères et la date du jour.
* @author moi
*@author http://www.lacl.u-pec/moi/
*@version 0.0  */
public class BonjourDate {
/** Unique point d'entrée de la classe et de l'application
*@param args tableau de paramètres sous forme de chaînes de caractères
*@return Pas de valeur de retour
*@exception exceptions Pas d'exceptions émises */
public static void main(String[] args) {
System.out.println("Bonjour, aujourd'hui: ");
System.out.println(new Date());}}
```

suite

- index.html
- allclasses-frame.html
- allclasses-noframe.html
- constant-values.html
- deprecated-list.html
- BonjourDate.html
- help-doc.html
- index-all.html
- overview-tree.html
- package-frame.html
- package-summary.html
- package-tree.html
- package-list
- stylesheet.css

Chapitre VIII – Exceptions

Chapitre VIII – Exceptions

Chapitre VIII – Exceptions

Chapitre VIII – Exceptions

Chapitre VIII – Exceptions

Qu'est-ce qu'une exception

En Java, une exception :

- est un objet, instance d'une classe d'exception
- est créee par un erreur
- contient des informations sur cette erreur
- provoque la sortie d'une méthode
- se propage de méthodes en méthodes
- peut être capturée et traitée
- peut provoquer l'arrêt du programme si aucun traitement.

Qu'est-ce qu'une exception

En Java, une exception :

- est un objet, instance d'une classe d'exception
- est créee par un erreur
- contient des informations sur cette erreur
- provoque la sortie d'une méthode
- se propage de méthodes en méthodes
- peut être capturée et traitée
- peut provoquer l'arrêt du programme si aucun traitement.

Qu'est-ce qu'une exception

En Java, une exception :

- est un objet, instance d'une classe d'exception
- est créee par un erreur
- contient des informations sur cette erreur
- provoque la sortie d'une méthode
- se propage de méthodes en méthodes
- peut être capturée et traitée
- peut provoquer l'arrêt du programme si aucun traitement.

Qu'est-ce qu'une exception

En Java, une exception :

- est un objet, instance d'une classe d'exception
- est créee par un erreur
- contient des informations sur cette erreur
- provoque la sortie d'une méthode
- se propage de méthodes en méthodes
- peut être capturée et traitée
- peut provoquer l'arrêt du programme si aucun traitement.

Qu'est-ce qu'une exception

En Java, une exception :

- est un objet, instance d'une classe d'exception
- est créee par un erreur
- contient des informations sur cette erreur
- provoque la sortie d'une méthode
- se propage de méthodes en méthodes
- peut être capturée et traitée
- peut provoquer l'arrêt du programme si aucun traitement.

Qu'est-ce qu'une exception

En Java, une exception :

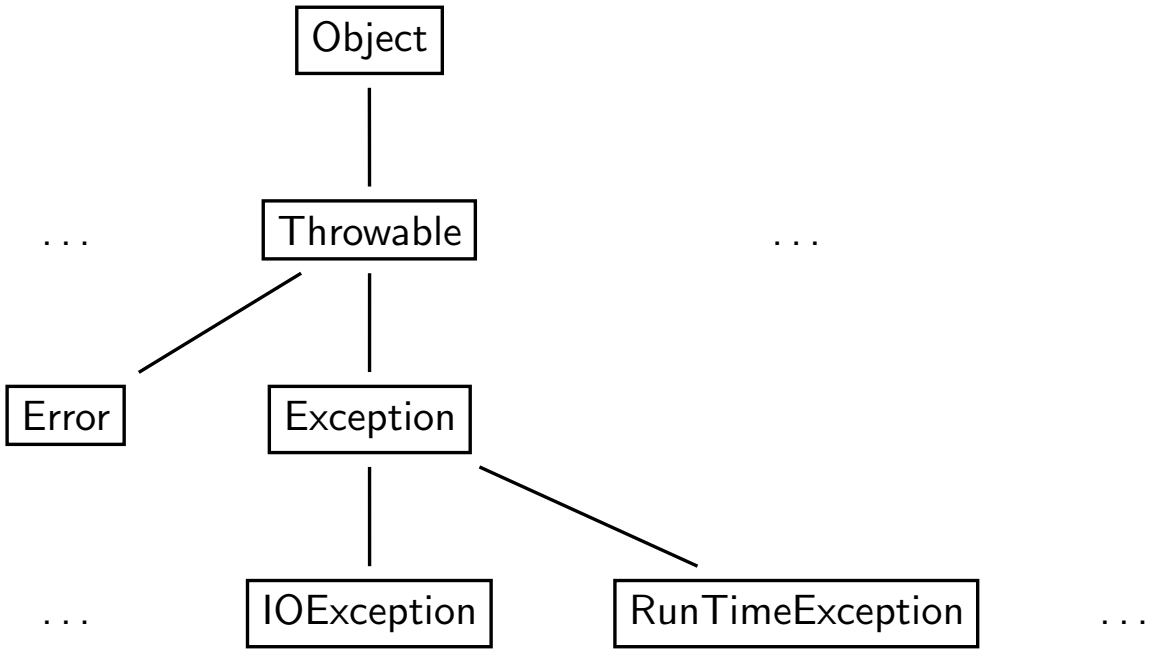
- est un objet, instance d'une classe d'exception
- est créee par un erreur
- contient des informations sur cette erreur
- provoque la sortie d'une méthode
- se propage de méthodes en méthodes
- peut être capturée et traitée
- peut provoquer l'arrêt du programme si aucun traitement.

Qu'est-ce qu'une exception

En Java, une exception :

- est un objet, instance d'une classe d'exception
- est créee par un erreur
- contient des informations sur cette erreur
- provoque la sortie d'une méthode
- se propage de méthodes en méthodes
- peut être capturée et traitée
- peut provoquer l'arrêt du programme si aucun traitement.

Exceptions prédéfinies



Exceptions prédéfinies

La classe Error contient les erreurs graves : StackOverflowError, ...
Les erreurs les plus souvent lancées font parties de la classe RunTimeException :

- ArithmeticException
- IllegalArgumentException
- ClassCastException
- IndexOutOfBoundException
- NullPointerException
- ...

On y trouve des erreurs prédéfinies :

- Division par zéro pour les entiers : ArithmeticException
- Référence nulle : NullPointerException
- Tentative de forçage de type illégale : ClassCastException
- Tentative de création d'un tableau de taille négative : NegativeArraySizeException
- Dépassement de limite d'un tableau : ArrayIndexOutOfBoundsException

exemple

```
public class Addition{
public static void main (String[] args){
int somme = 0 ;
for (int ix = 0 ; ix < args.length; ix++) {
somme += Integer.parseInt(args[ix]) ;}
System.out.println("somme=" + somme);} // main
}
```

```
 java Addition 1 2 trois 4
Exception in thread "main" java.lang.NumberFormatException:
For input String "trois"
at
java.lang.NumberFormatException.forInputString(NumberFormatExcept
at java.lang.Integer.parseInt(Integer.java:481)
at java.lang.Integer.parseInt(Integer.java:514)
at Addition.main(Addition.java:7)
```

Définir ses propres exceptions

Règles : Pour définir ses propres exceptions on crée une sous-classe de la classe Exception.

```
class MonException extends Exception { ...}
```

Lancer une exception

Règles :

- créer une instance de la classe d'exception que l'on veut lancer
- lancer une exception avec le mot clé throw

```
throw new MonException() ;
```

Si cette instruction est exécutée alors l'exécution normale du programme s'arrête :
▷ soit l'exception lancée est attrapée dans une des méthodes appelant le bloc en cours
▷ soit on sort du programme

Règle : Java impose de déclarer qu'une méthode peut lancer une exception sans l'attraper sauf pour les exceptions des classes **Error** et **RuntimeException** avec la clause throws.

```
 public class MonException extends Exception}{
 MonException(){System.out.println("ceci est une erreur");}
 }
public class EssaiException {
public static void main throws MonException  (String[] args){
System.out.println("message 1");
if (true) throw new MonException() ;
System.out.println("message 2");
System.out.println("message 3");
```

exécution :

```
message 1
ceci est une erreur
Exception in thread "main" EssaiException at
EssaiException.main(EssaiException.java: 4)
```

Attraper une exception

Règle : Pour attraper une exception on utilise deux blocs consécutifs

- try {...} : contient l'instruction throw
- catch(MonException e) {...} : contient les instructions à exécuter si le bloc try a lancé une instance de MonException

```
try {
suiteInstruc1;
...throw new MonException()...;
suiteInstruc2; }
catch(MonException e) {suiteInstr3;}
suiteInstruc4;
```

throw n'est pas exécuté : suiteInstruc1, suiteInstruc2, suiteInstruc4 sont successivement exécutées

throw est exécuté : suiteInstruc1, suiteInstruc3, suiteInstruc4 sont successivement exécutées

une exception d'un autre type est lancée : message d'erreur ou traitement par un bloc appelant

Exemple

```
public class Addition{
public static void main (String[] args){
int somme = 0 ;
for (int ix = 0 ; ix < args.length; ix++) {
try{int k = Integer.parseInt(args[ix]);
     somme +=  k;}
catch(Exception e){} }
System.out.println("somme=" + somme);} // main
}
```

exécution :

```
java Addition 1 2 trois 4
somme = 7
```

Attraper plusieurs exceptions

On peut enchaîner les blocs `catch` si le bloc `try` est suscceptible de lancer plusieurs types d'exception.
On peut imbriquer un bloc `try catch` dans un bloc `try`

Bloc finally

Règle : un bloc try est nécessairement suivi soit d'un bloc finally soit d'un ou plusieurs blocs catch suivis éventuellement d'un bloc finally.
Le bloc `finally` contient des instructions qui seront exécutées à la sortie du bloc `try` qui le précède immédiatement ou bien après l'exécution du bloc `catch` intercalé entre les blocs `try` et `finally`.
Le bloc `finally` est alors exécuté juste après la sortie du bloc `try` s'il n'y a pas de catch ou juste après l'exécution du bloc `catch` intercalé.
On a donc soit

- **try** + **finally**
- **try** + **catch**
- **try** + **catch** + **finally**

Propagation d'une exception

Si une méthode mB est appelée par une méthode mA et si mB lance une exception sans l'attraper alors la méthode mA peut attraper l'exception lancée par mB : il y a propagation de l'exception.
Le code doit suivre le schéma de l'exemple suivant :

```
class ExceptPropag {
static void methodeNiveauC () throws MonException {
try {
if (true) throw new MonException() ;
System.out.println(" ? ? ? C") ;
    }
finally {
System.out.println("niveau C") ;
}}
static void methodeNiveauB ()  throws MonException {
try {
methodeNiveauC() ;
System.out.println( " ? ? ? B ") ;
    }
finally {
System.out.println("niveau B") ;
}}
static void methodeNiveauA () {
try {
methodeNiveauB() ;
 System.out.println( " ? ? ? A ") ;
   }
catch(MonException e){
System.out.println( "exception attrape au niveau A") ;
e.printStackTrace() ;
}
System.out.println( "on reprend comme si de rien était") ;} }
```

Exécution de `methodeNiveauA()` dans une classe `EssaiExceptPropag` :

```
ceci est une erreur
niveau C
niveau B
exception attrapée au niveau A
MonException
        : at ExceptPropag.methodeNiveauC (ExceptPropag. java :
4)
        : at ExceptPropag.methodeNiveauB (ExceptPropag. java :
13)
        : at ExceptPropag.methodeNiveauA (ExceptPropag. java :
22)
        : at EssaiExceptPropag.main (EssaiExceptPropag.java : 3 )
on reprend comme si de rien n'etait
```

(voir méthodes de la classe Throwable : printStackTrace() ...)

Exécution de `methodeNiveauA()` dans une classe `EssaiExceptPropag` :

ceci est une erreur
niveau C
niveau B
exception attrapée au niveau A
MonException
: at ExceptPropag.methodeNiveauC (ExceptPropag. java : 4)
: at ExceptPropag.methodeNiveauB (ExceptPropag. java : 13)
: at ExceptPropag.methodeNiveauA (ExceptPropag. java : 22)
: at EssaiExceptPropag.main (EssaiExceptPropag.java : 3)
on reprend comme si de rien n'etait

(voir méthodes de la classe Throwable : printStackTrace() ...)

Exécution de `methodeNiveauA()` dans une classe `EssaiExceptPropag` :

ceci est une erreur
niveau C
niveau B
exception attrapée au niveau A
MonException
: at ExceptPropag.methodeNiveauC (ExceptPropag. java : 4)
: at ExceptPropag.methodeNiveauB (ExceptPropag. java : 13)
: at ExceptPropag.methodeNiveauA (ExceptPropag. java : 22)
: at EssaiExceptPropag.main (EssaiExceptPropag.java : 3)
on reprend comme si de rien n'etait

(voir méthodes de la classe Throwable : printStackTrace() ...)

Exécution de `methodeNiveauA()` dans une classe `EssaiExceptPropag` :

ceci est une erreur
niveau C
niveau B
exception attrapée au niveau A
MonException
: at ExceptPropag.methodeNiveauC (ExceptPropag. java : 4)
: at ExceptPropag.methodeNiveauB (ExceptPropag. java : 13)
: at ExceptPropag.methodeNiveauA (ExceptPropag. java : 22)
: at EssaiExceptPropag.main (EssaiExceptPropag.java : 3)
on reprend comme si de rien n'etait

(voir méthodes de la classe Throwable : printStackTrace() ...)

Exécution de `methodeNiveauA()` dans une classe `EssaiExceptPropag` :

ceci est une erreur
niveau C
niveau B
exception attrapée au niveau A
MonException
: at ExceptPropag.methodeNiveauC (ExceptPropag. java : 4)
: at ExceptPropag.methodeNiveauB (ExceptPropag. java : 13)
: at ExceptPropag.methodeNiveauA (ExceptPropag. java : 22)
: at EssaiExceptPropag.main (EssaiExceptPropag.java : 3)
on reprend comme si de rien n'etait

(voir méthodes de la classe Throwable : printStackTrace() ...)

Exécution de `methodeNiveauA()` dans une classe `EssaiExceptPropag` :

ceci est une erreur
niveau C
niveau B
exception attrapée au niveau A
MonException
: at ExceptPropag.methodeNiveauC (ExceptPropag. java : 4)
: at ExceptPropag.methodeNiveauB (ExceptPropag. java : 13)
: at ExceptPropag.methodeNiveauA (ExceptPropag. java : 22)
: at EssaiExceptPropag.main (EssaiExceptPropag.java : 3)
on reprend comme si de rien n'etait

(voir méthodes de la classe Throwable : printStackTrace() ...)

Exécution de `methodeNiveauA()` dans une classe `EssaiExceptPropag` :

```
ceci est une erreur
niveau C
niveau B
exception attrapée au niveau A
MonException
        : at ExceptPropag.methodeNiveauC (ExceptPropag. java :
4)
        : at ExceptPropag.methodeNiveauB (ExceptPropag. java :
13)
        : at ExceptPropag.methodeNiveauA (ExceptPropag. java :
22)
        : at EssaiExceptPropag.main (EssaiExceptPropag.java : 3 )
on reprend comme si de rien n'etait
```

(voir méthodes de la classe Throwable : printStackTrace() ...)

Exécution de `methodeNiveauA()` dans une classe `EssaiExceptPropag` :

ceci est une erreur
niveau C
niveau B
exception attrapée au niveau A
MonException
: at ExceptPropag.methodeNiveauC (ExceptPropag. java : 4)
: at ExceptPropag.methodeNiveauB (ExceptPropag. java : 13)
: at ExceptPropag.methodeNiveauA (ExceptPropag. java : 22)
: at EssaiExceptPropag.main (EssaiExceptPropag.java : 3)
on reprend comme si de rien n'etait

(voir méthodes de la classe Throwable : printStackTrace() ...)

Chapitre IX – Types énumérés

Chapitre IX – Types énumérés

Chapitre IX – Types énumérés

Définition

Un *type énuméré* est un type qui permet de définir une variable comme élément d'un ensemble fini de valeurs constantes.
Les classes qui implémentent ces types diffèrent des autres classes dans leur déclaration : leur en-tête doit commencer par `enum` au lieu de `class`.
De plus, les identificateurs des champs d'un type énuméré sont écrits en majuscule car ce sont des constantes.
Chaque élément d'une énumération est un objet à part entière.

Exemple

```
public enum ArcEnCiel{
ROUGE, ORANGE, JAUNE, VERT, BLEU, INDIGO, VIOLET
}

public class EssaiArcEnCiel{
public static void main(String[] arg) {
ArcEnCiel aec =  ArcEnCiel.valueOf(arg[0]);
if(aec.toString() == "ROUGE") System.out.println("chaud");
else if(aec==ArcEnCiel.BLEU)) System.out.println("froid");
System.out.println(aec.name());
System.out.println(aec.ordinal());
}
// toString() renvoie la chaîne de caractères identifiant la constante dans l'énumération
//ArcEnCiel.BLEU est une constante (statique) de type ArcenCiel
}
```

A l'exécution de `java EssaiArcEnCiel BLEU` on obtient `froid bleu 4`

`java.lang.Enum`

Règle : toute classe `enum` hérite automatiquement de la classe `java.lang.Enum`.
Par conséquent, tout type énuméré ne peut étendre aucune autre classe.
Toute classe `enum` hérite en particulier des méthodes suivantes :

- `toString()` : renvoie le nom de l'objet défini dans l'énumération
- `equals()` : teste l'égalité
- `valueOf(String s)` : méthode statique finale qui renvoie un objet du type énuméré dont le nom correspond à la chaîne s
- `values()` : méthode statique finale qui renvoie un tableau des valeurs énumérées dans l'ordre d'énumération
- `ordinal()` : méthode finale qui renvoie le numéro d'ordre dans l'énumération en commençant par 0
- `name()` : méthode finale qui renvoie la chaîne de caractères du nom de l'objet

Utilisation

```
switch (aec) {
case ROUGE: case ORANGE: case JAUNE :
System.out.println("couleur chaude"); break ;
case VERT : case BLEU : case INDIGO : case VIOLET :
System.out.println("couleur froide"); break;
default : System.out.println("???"); break;
```

On peut tester un objet de type énuméré comme un entier avec `switch`

Utilisation

```
for (ArcEnCiel a : ArcEnCiel.values()){System.out.println(a.name());}
```

On peut itérer sur toutes les valeurs du type énuméré de `EnumClasse` par
`for (EnumClasse e : EnumClasse.values())`
la variable e prend successivement toutes les valeurs énumérées dans l'ordre d'énumération.

Ajout d'informations

Il est possible d'ajouter des informations propres à chaque valeur énumérée sous forme d'attributs.
On doit alors écrire constructeurs et méthodes correspondant à ces attributs.

Exemple

```
public enum ArcEnCiel2{
ROUGE(6.2E-7,"chaud"),// 2 attributs double et String
ORANGE(5.8E-7,"chaud"),
JAUNE(5.6E-7,"chaud"),
VERT(5.0E-7,"froid"),
BLEU(4.5E-7,"froid"),
INDIGO(4.0E-7,"froid"),
VIOLET(3.8E-7,"froid");
private final double l;
private final String chaleur;
private ArcEnCiel2(double d, String s){this.l=d;this.chaleur = s;}
public static String couleur(double d){
String s = "invisible";
if(d>ROUGE.l || d<VIOLET.l) return s;
for (ArcEnCiel2 a : ArcEnCiel2.values()){
if (d>a.l)
return( "couleur " + a.chaleur + " entre " + ArcEnCiel2.values()[a.ordinal()-1].name() +
" et "+a.name()) +;}
return s;                          }}
public class EssaiArcEnCiel2{
public static void main(String[] arg) {
double d= Double.parseDouble(arg[0]);
System.out.println(ArcEnCiel2.couleur(d));
}}
```