

Exceptions

Java Standard Edition





Exceptions

Objectifs du cours

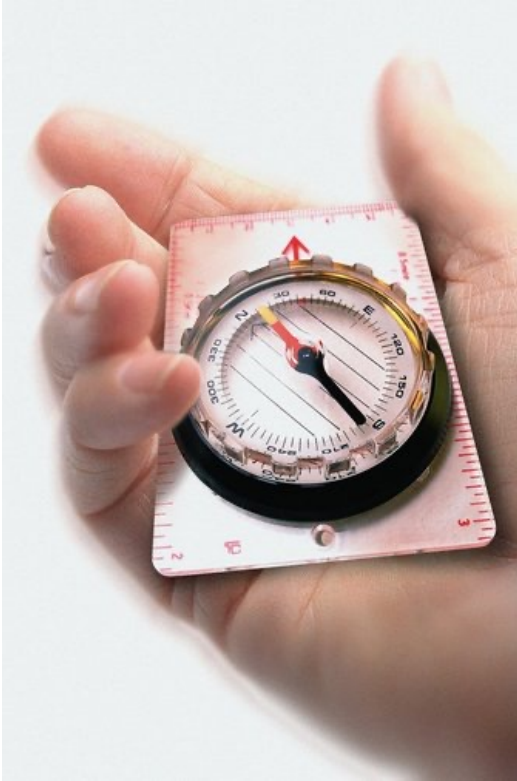
En complétant ce cours, vous serez en mesure de :

- **Expliquer** ce que sont les exceptions
- **Les Gérer** et ainsi garder votre code clair et efficace
- **Créer** et **lancer** vos propres exceptions
- **Utiliser** les exceptions et rendre votre code plus sûr



Exceptions

Plan de cours



- **Introduction.** Que sont les exceptions ?
- **Les blocs try/catch/finally.** Plusieurs façons de les manipuler.
- **Les mots-clés throw/throws.** Centraliser la gestion et lever les exceptions.
- **Créer vos propres exceptions.** Que faire pour redéfinir le processus ?
- **Lire une trace de pile.** Comment comprendre un Stack Trace.

Exceptions

INTRODUCTION

Comment gérer les situations anormales en Java ?





Exceptions

Définition

"Une exception est un événement qui se produit pendant l'exécution d'un programme et qui perturbe le flux normal des instructions du programme."

– Sun Microsystems



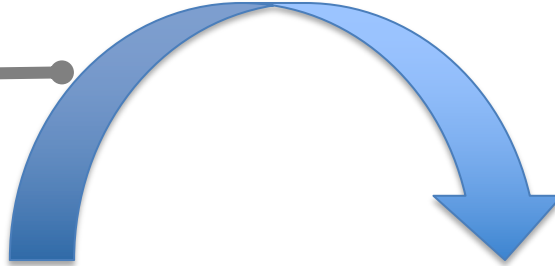
Qu'est-ce qu'une exception ?

- **Introduction**

Une exception
est levée !

```
...  
public void doSomething() {  
    doSomeStuff();  
    doSomeOtherStuff();  
    doItAgain();  
}  
...
```

Le programme s'exécute
bien mais...



Exception
Ceci est une exception

Une erreur ?
Non, ce n'est **PAS**
nécessairement une
erreur



Sortes
Il existe deux sortes
d'exceptions (on
verra plus tard)



Qu'est-ce qu'une exception ?

- Les exceptions sont des objets Java comme les autres
- Ce sont toutes des sous-classes de **Throwable**



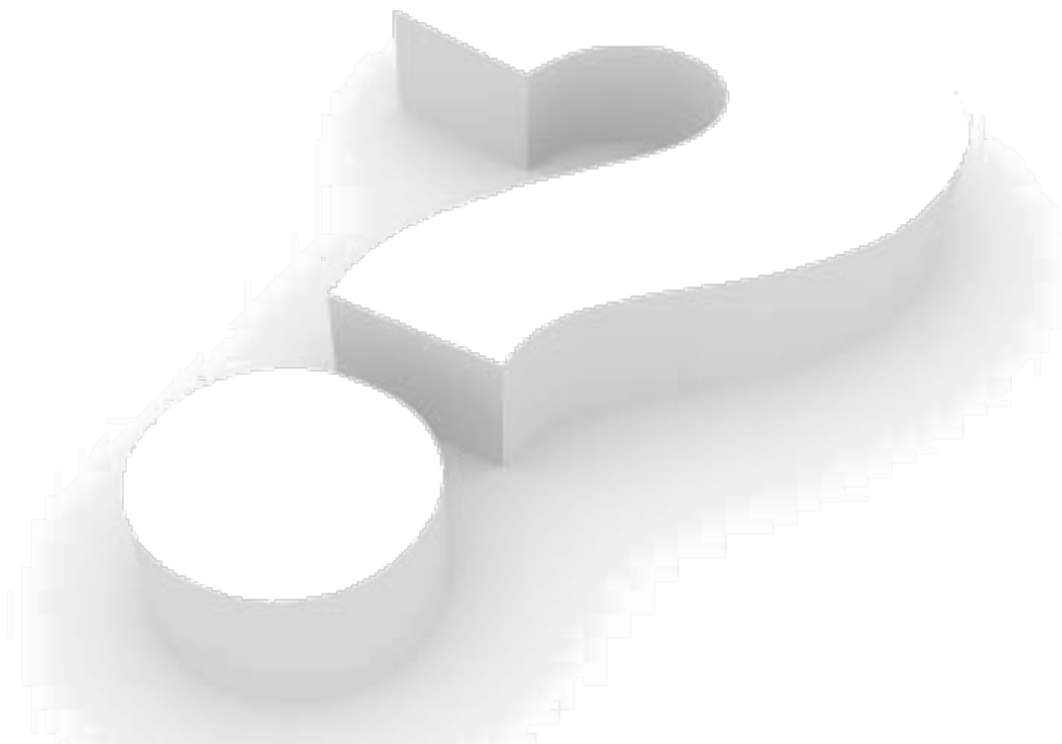


Pourquoi utiliser la gestion des exceptions ?

- Seriez-vous content si votre programme s'arrête à la première exception ?
 - Une calculatrice plante lorsque vous divisez par 0 ?
 - Votre téléphone explose lorsque vous entrez un mauvais numéro ?
 - Les ordinateurs affichent toujours un écran bleu ? (certains oui mais on ne peut rien faire pour eux...)
- La gestion des exceptions sépare le code de l'application de sa gestion



Questions ?





Exercises

- Faites planter votre programme Game of Life ! 😊
 - Essayez ces deux manières simples :
 - La première :
 - Dans votre méthode principale, passez des valeurs de taille négatives au constructeur de la classe World
 - Le deuxième :
 - Dans votre méthode principale, passez une valeur nulle au constructeur prenant un tableau bidimensionnel en paramètre
- Pour chacun, exécutez votre application et expliquez ce qui s'est passé

Exceptions

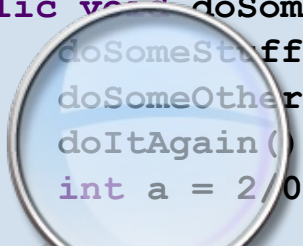
LES BLOCS TRY/CATCH/FINALLY

Plusieurs façons de les gérer




Le bloc try

- Le bloc **try** est utilisé pour encapsuler du code risqué :
 - Cela pourrait déclencher (**throw**) une exception !



```
public void doSomething() {  
    doSomeStuff();  
    doSomeOtherStuff();  
    doItAgain();  
    int a = 2/0;  
}
```



```
public void doSomething() {  
    doSomeStuff();  
    doSomeOtherStuff();  
    doItAgain();  
    try {  
        int a = 2/0;  
    } ...  
}
```



Les blocs try/catch/finally

Le bloc catch

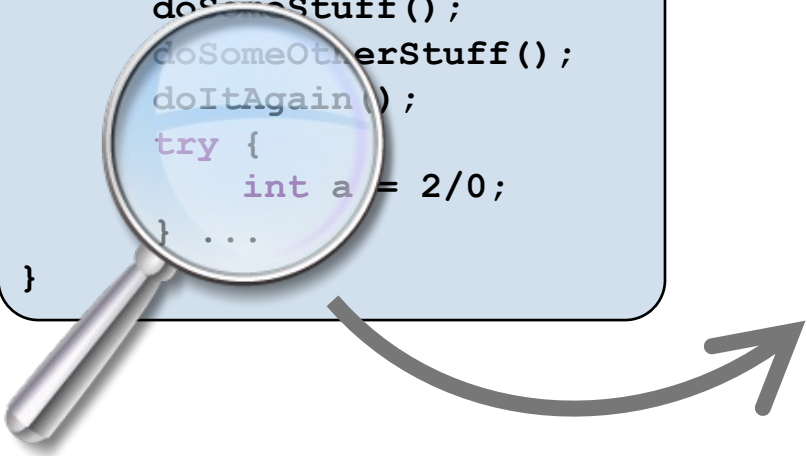
- Le bloc **try** n'est **JAMAIS** seul :
 - Soit suivi d'un ou plusieurs blocs **catch**, soit d'un bloc **finally**, soit les deux
- Le bloc **catch** est utilisé pour gérer l'exception
 - Contient ce qu'il faut faire au lieu de crasher



Le bloc catch

- Les blocs try/catch/finally

```
public void doSomething() {  
    doSomeStuff();  
    doSomeOtherStuff();  
    doItAgain();  
    try {  
        int a = 2/0;  
    } ...  
}
```



```
public void doSomething() {  
    doSomeStuff();  
    doSomeOtherStuff();  
    doItAgain();  
    try {  
        int a = 2/0;  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```



Le bloc **finally**

- Le bloc **finally** est exécuté quelle que soit l'exception levée ou non

```
try {  
    doSomeStuff();  
    doOtherStuff();  
} catch (IOException ioe) {  
    doHandleIOException();  
} finally {  
    doSomethingInAnyCase();  
}
```

```
try {  
    doSomeStuff();  
    doOtherStuff();  
} finally {  
    doSomethingInAnyCase();  
}
```



Manipulation simple

• Essaye ça ! Les blocs

Utilisez-le lorsque des exceptions peuvent se produire.

Attrape le !
Utilisez-le avec un bloc **try** pour gérer les exceptions.

```
try {  
    doSomeStuff();  
} catch (Exception e) {  
    doHandleException();  
}
```

Du code

Ce code peut lever une exception.

Exception e

Le type et le nom de l'exception que nous pouvons intercepter.

Manipulez-le !

Du code à exécuter lorsqu'une exception se produit



Manipulation complexe (avant Java 7)

- Les blocs try/

Du code

Ce code peut lever une exception.

Encore du code

Ce code ne sera pas exécuté si `doSomeStuff()` lève une exception.

```
try {  
    doSomeStuff();  
    doOtherStuff();  
} catch (IOException ioe) {  
    doHandleIOException();  
} catch (Exception e) {  
    doHandleException();  
}
```

IOException ioe
N'attrape que les IOExceptions et les sous-classes

Exception e
Attrape toutes les autres exceptions



Manipulation complexe (depuis Java 7)

- Java 7 introduit la notation multi-catch :

```
try {  
  
    doSomeStuff();  
    doOtherStuff();  
  
} catch (IOException|SQLException ex) {  
  
    doHandleException();  
  
}
```



Les blocs try/catch/finally

Deux types d'exceptions

- Les exceptions vérificatrices :
 - Le programmeur doit les anticiper
 - Le programme ne doit pas cracher
 - Vous devez écrire du code au cas où ils apparaîtraient
 - Si vous ne le faites pas, le programme ne compilera pas
- Exemple : essayer de charger un fichier qui n'existe peut-être pas





Les blocs try/catch/finally

Deux types d'exceptions

- Les exceptions d'exécution :
 - Exceptions qui étendent la classe **RuntimeException**
 - Exceptions causées par l'état interne d'un programme
 - Ne peut pas être prédit (dans la plupart des cas)
 - Dans certains cas, il vaut mieux les empêcher d'apparaître que de les gérer
- Exemple : vous essayez de lire une valeur hors des limites du tableau





Quizz

- Remplissez les blancs :

Les exceptions peuvent être de **2** sortes.

Pour gérer une exception, il FAUT utiliser le mot-clé **try**

Ensuite, il faut ajouter au moins un des mots-clés suivants :
catch or finally

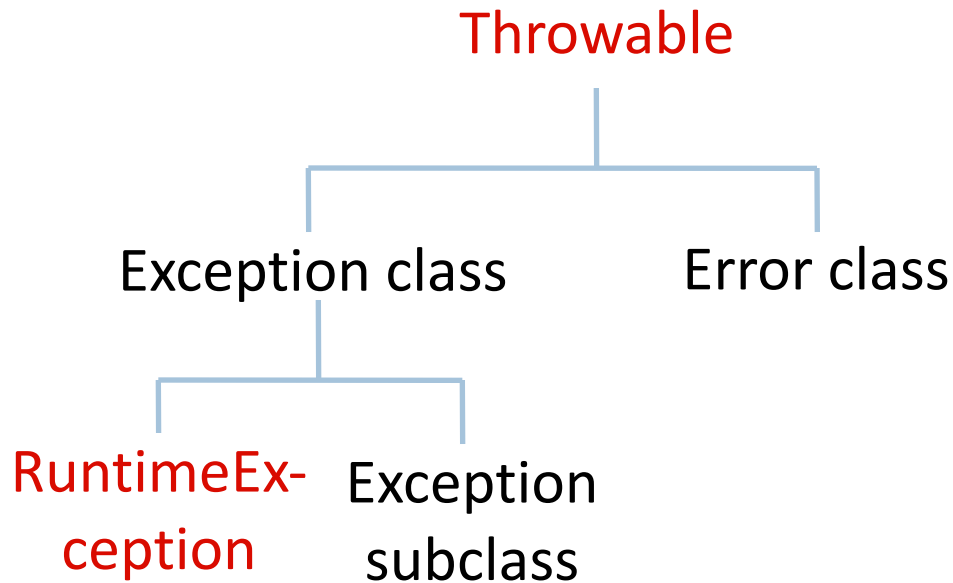
Vous pouvez utiliser plusieurs blocs **catch** dans une seule gestion des exceptions.

Le bloc **finally** vous assure que ses instructions seront exécutées quoi qu'il arrive.



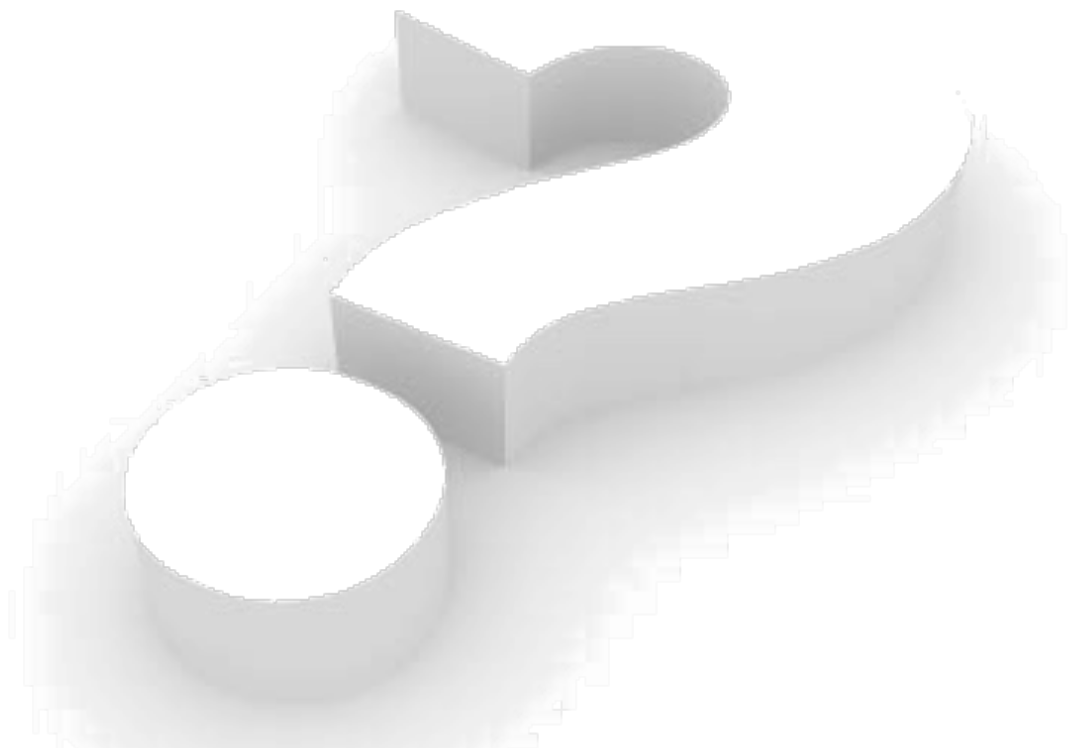
Quizz

- Pouvez-vous remplacer le nom de la classe ici ?





Questions ?



Exceptions

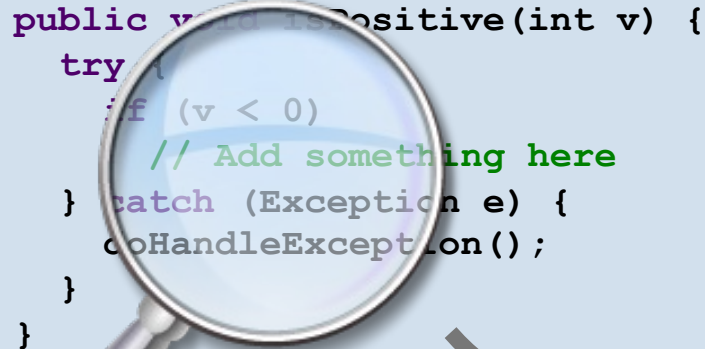
LES MOTS-CLÉS THROW/THROWS

Centraliser la gestion et lever une exception



Le mot-clé throw

- Le mot-clé **throw** est utilisé pour... lever une exception quand vous le souhaitez



```
public void isPositive(int v) {  
    try {  
        if (v < 0)  
            // Add something here  
    } catch (Exception e) {  
        doHandleException();  
    }  
}
```

```
public void isPositive(int v) {  
    try {  
        if (v < 0)  
            throw new  
                Exception("negative value");  
    } catch (Exception e) {  
        doHandleException();  
    }  
}
```



Les blocs try/catch/finally

Le mot-clé **throws**

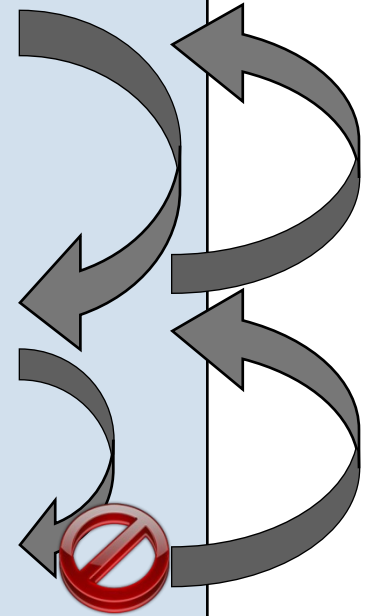
- Le mot clé **throws** est utilisé pour indiquer que la méthode ne gère pas l'exception localement
- Utile pour créer un système global de gestion des erreurs



Le mot-clé throws

- Les

```
public void methode1() {  
    try {  
        methode2();  
    } catch (IOException ioe) {  
        System.err.println(ioe.getMessage());  
    }  
}  
  
public void methode2() throws IOException {  
    methode3();  
}  
  
public void methode3() throws IOException {  
    throw new IOException("File missing");  
}
```





Les blocs try/catch/finally

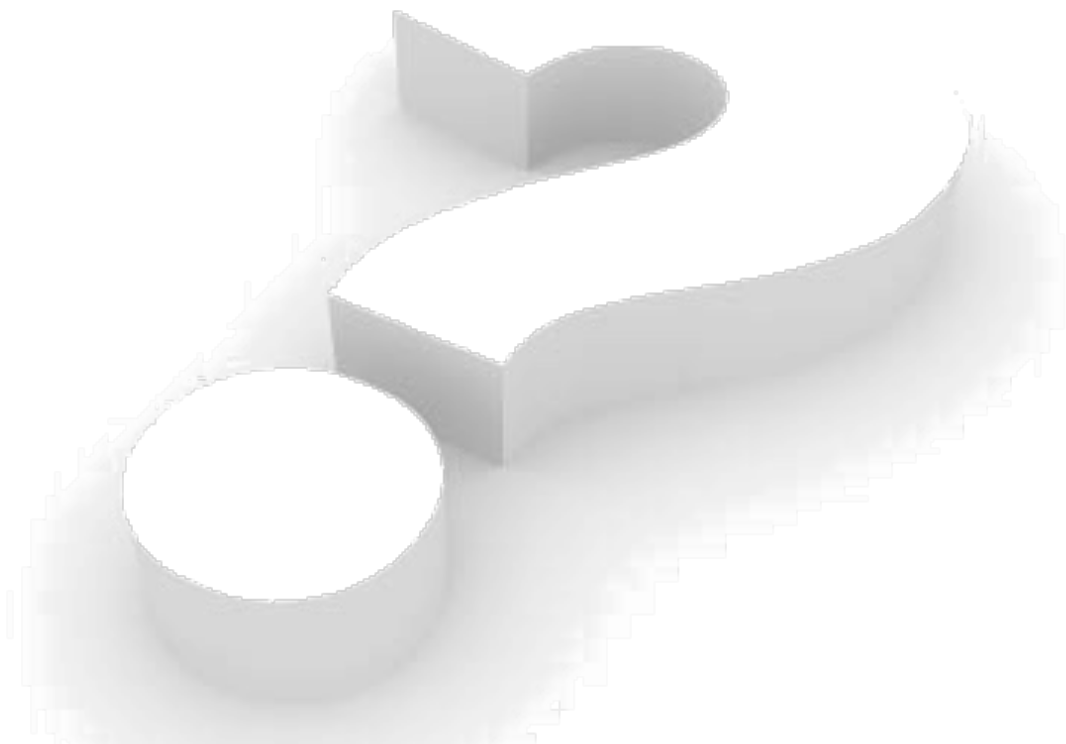
throw/throws

- Le mot clé **throw** est toujours placé à l'intérieur d'une méthode
- Le mot clé **throws** est toujours placé sur la déclaration de la méthode

```
public void methode3() throws IOException {  
    throw new IOException("File missing");  
}
```



Questions ?





Exercises (1/3)

- Mettez à jour votre méthode main :
 - Au début de l'exécution, le programme doit demander à l'utilisateur d'entrer le nombre de colonnes et de lignes qu'il souhaite pour son monde
 - Utilisez la classe Scanner pour récupérer l'entrée utilisateur (consultez la Javadoc pour plus d'informations)

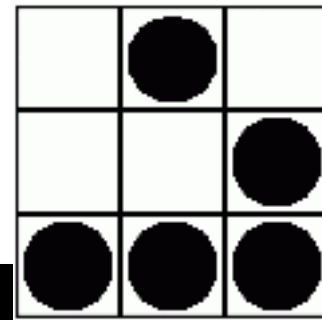
```
Please enter the number of columns of the world:  
10  
Please enter the number of rows of the world:  
10
```



Les blocs try/catch/finally

Exercises (2/3)

- Exécutez votre programme et entrez une mauvaise valeur lorsqu'il vous demande d'entrer un numéro de colonne ou de ligne.
- Vous devez voir une jolie exception...
- Utilisez le bloc **try / catch** pour gérer les mauvaises entrées d'utilisateurs sans arrêter votre programme !





Les blocs try/catch/finally

Exercises (3/3)

- Lorsque de mauvaises valeurs sont passées aux constructeurs World, l'exception qui est levée n'est pas très explicite... Pourquoi `NegativeArraySizeException` ?
 - L'utilisateur de la classe doit regarder à l'intérieur de la classe World pour bien comprendre où se situe le problème...
- Améliorer la clarté des exceptions :
 - Mettre à jour les constructeurs de classe World :
 - Vérifier que les valeurs reçues en paramètres sont valides.
 - Si ce n'est pas le cas, lancez une `InvalidArgumentException` avec un message explicatif.

Exceptions

CRÉEZ VOS PROPRES EXCEPTIONS

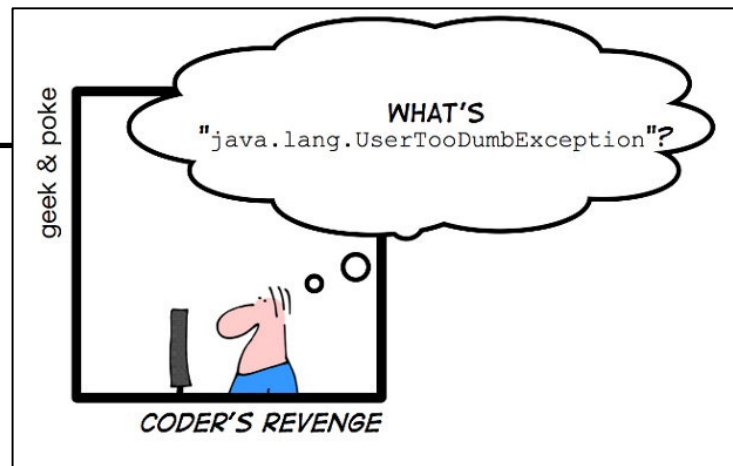


Créez vos propres exceptions

Créer une exception

- faire hériter de :
 - **Exception** si vous souhaitez créer une exception vérificatrice
 - **RuntimeException** sinon

```
public class MyException extends Exception {  
    // This is valid exception !  
}
```





Créez vos propres exceptions

Créer une exception

- Définissez votre propre logique dans le constructeur
 - Évidemment, vous pouvez également étendre une exception existante

```
public class NegativeNumberException extends NumberException
{

    public NegativeNumberException(int num) {
        super("The number "+num+" is negative");
    }

    // This is valid exception too !

}
```



Créez vos propres exceptions

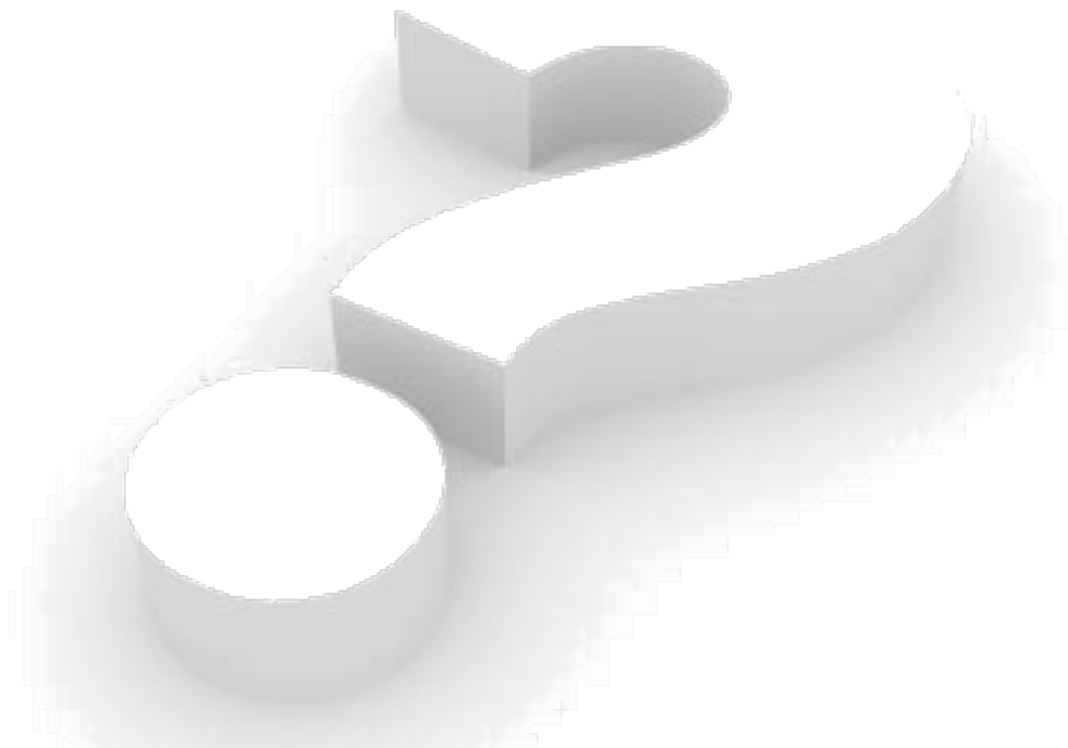
L'Utiliser

- Comme toutes les autres exceptions :

```
public void throw1() throws MyException {  
    throw new MyException();  
}  
  
public void throw2() {  
    try {  
        // Do some stuff  
        if (a == -2) throw new NegativeNumberException(a);  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```



Questions ?



Exceptions

LIRE UNE STACKTRACE D'EXCEPTION

Bien comprendre la cause d'une exception.



Présentation

- Une trace de pile est un vidage de la pile d'exécution actuelle
 - Affiche les appels de méthode exécutés sur ce thread de bas en haut
 - Les exceptions gardent une Stack Trace du moment où elles sont levées
- Vous pouvez l'afficher dans le flux d'erreur standard grâce à la méthode d'instance Exception :
`void printStackTrace()`



Présentation

- Exception exemple:
 - Ici, l'exception a été levée dans la fonction **mash()** de la classe **MyClass**, à la ligne 9
 - Cette fonction **mash()** a été appelée par la fonction **crunch()**
 - Et la fonction **crunch()** a été appelée dans le **main()**

```
java.lang.NullPointerException
    at MyClass.mash(MyClass.java:9)
    at MyClass.crunch(MyClass.java:6)
    at MyClass.main(MyClass.java:3)
```




Lire une stacktrace d'exception

Cause d'Exception

- Lorsqu'une Exception est levée, cela peut être dû à une autre Exception
 - Cette autre Exception est appelée **cause**
- La classe Exception fournit deux constructeurs prenant une instance Throwable comme cause



Lire une stacktrace d'exception

Cause d'Exception

- Stack Trace d'une exception avec une cause :

```
HighLevelException: MidLevelException: LowLevelException
    at Junk.a(Junk.java:13)
    at Junk.main(Junk.java:4)
Caused by: MidLevelException: LowLevelException
    at Junk.c(Junk.java:23)
    at Junk.b(Junk.java:17)
    at Junk.a(Junk.java:11)
    ... 1 more
Caused by: LowLevelException
    at Junk.e(Junk.java:30)
    at Junk.d(Junk.java:27)
    at Junk.c(Junk.java:21) ... 3 more
```

Quiz 1/3

```
public class ComplexStack {
    public void saveUser(String firstName, String
        lastName, String email) throws
        UserPersistException {
        try {
            addUserInDB(firstName, lastName, email);
        } catch (DBConnexionException e) {
            throw new UserPersistException("Error
                persisting user.", e);
        }
    }

    private Connection connectToDB()
        throws SQLException {
        return DriverManager.getConnection(URL, USER,
            PASSWORD);
    }
    ...
}
```

Quiz 2/3

```
private void addUserInDB(String firstName,
                          String lastName, String mail)
    throws DBConnexionException {
    try {
        Connection connection = connectToDB();
    } catch (SQLException e) {
        throw new DBConnexionException("Can't
                                         connect to the DB.", e);
    } //...
}

public static void main(String[] args) {
    ComplexStack cs = new ComplexStack();
    try {
        cs.saveUser("John", "Doe", "john@doe.fr");
    } catch (UserPersistException e) {
        e.printStackTrace();
    }
}
}
```



Lire une stacktrace d'exception

Quizz (3/3)

Trouvez la raison de l'exception en lisant ce Stack Trace

com.cci.trace.UserPersistException: Error persisting user.

at com.cci.trace.ComplexStacktrace.saveUser(ComplexStacktrace.java:14)

at com.cci.trace.ComplexStacktrace.main(ComplexStacktrace.java:34)

Caused by: com.cci.trace.DBConnectionException: Can't connect to the DB.

at com.cci.trace.ComplexStacktrace.persistInDB(ComplexStacktrace.java:23)

at com.cci.trace.ComplexStacktrace.saveUser(ComplexStacktrace.java:12)

... 1 more

Caused by: java.sql.SQLException: No suitable driver found for ...

at java.sql.DriverManager.getConnection(DriverManager.java:602)

at java.sql.DriverManager.getConnection(DriverManager.java:185)

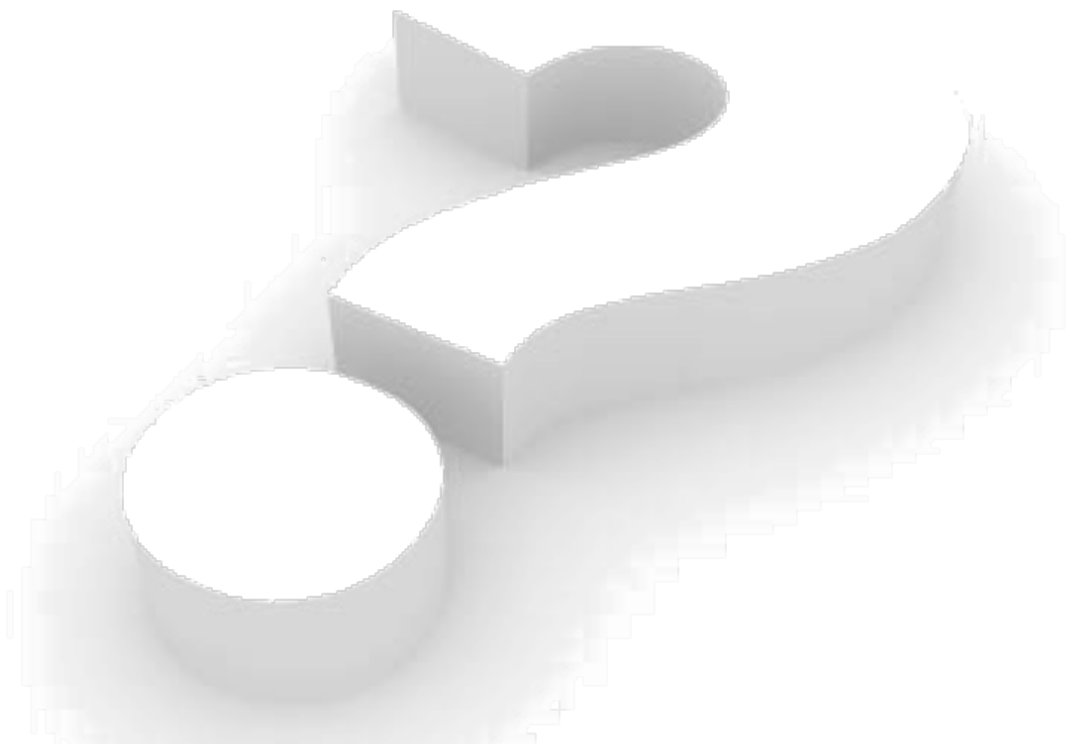
at com.cci.trace.ComplexStacktrace.openConnectionToDatabase(ComplexStacktrace.java:28)

at com.cci.trace.ComplexStacktrace.persistInDB(ComplexStacktrace.java:21)

... 2 more



Questions ?





Exceptions

Fin

Merci de votre attention