8. TP 6 : Gestion des Exceptions en Java

8.1 Introduction aux Exceptions

Les exceptions sont des événements qui surviennent lors de l'exécution d'un programme, interrompant son déroulement normal. En Java, les exceptions permettent de capturer ces erreurs et de les gérer de manière contrôlée sans provoquer l'arrêt brutal du programme.

8.1.1 La hiérarchie des Exceptions

Toutes les exceptions en Java sont des sous-classes de `java.lang.Throwable`. Voici la hiérarchie de base :

- `Throwable`
- `Error` (catégorie non récupérable, erreurs système)
- `Exception` (catégorie récupérable, à gérer explicitement)
- `RuntimeException` (sous-classe d'`Exception`, exceptions d'exécution non vérifiées)

8.2 Gestion des Exceptions en Java

La gestion des exceptions se fait avec quatre principaux blocs : `try`, `catch`, `finally` et `throw`.

try Permet d'entourer le code susceptible de générer une exception.
catch Capture et gère l'exception si elle survient dans le bloc `try`.

finally Exécute du code après le bloc `try` (et éventuellement `catch`), qu'une exception soit survenue ou non.

throw Utilisé pour lever une exception manuellement.

8.2.1 Exemple Simple de Gestion d'Exception

Voici un exemple simple de gestion d'une `ArithmeticException` :

```
public class ExceptionExample {
    public static void main(String[] args) {
        try {
            int result = 10 / 0; // Cette ligne va générer une
    ArithmeticException
        } catch (ArithmeticException e) {
            System.out.println("Erreur : Division par zéro !")
    ;
        } finally {
            System.out.println("Bloc finally exécuté.");
        }
    }
}
```

Dans cet exemple :

- Le bloc `try` contient du code susceptible de lever une exception (ici une `ArithmeticException`).
- Le bloc `catch` capture et gère cette exception.
- Le bloc `finally` est exécuté dans tous les cas, que l'exception soit levée ou non.

8.3 Exemple d'Application : Gestion de Comptes Bancaires

8.3.1 Classe `Compte` Sans Gestion d'Exception

```
public class Compte {
    private String numero;
    private float solde;

    public Compte(String numero, float solde) {
        this.numero = numero;
        this.solde = solde;
    }

    public void verser(float mt) {
        solde += mt;
    }

    public void retirer(float mt) {
        if (solde >= mt) {
            solde -= mt;
        } else {
```

```
            System.out.println("Solde insuffisant!");
        }
    }

    public float getSolde() {
        return solde;
    }

    @Override
    public String toString() {
        return "[" + numero + ", " + solde + "]";
    }
}
```

Sans gestion des exceptions, voici ce qui peut se produire :

— Si le solde est insuffisant, le programme affiche un message, mais les actions suivantes peuvent être fausses (ex : mise à jour incorrecte).

8.3.2 Amélioration avec Gestion d'Exception

Nous allons améliorer l'exemple en ajoutant la gestion d'une exception pour empêcher les actions incorrectes si un retrait échoue.

```
public class Compte {
    private String numero;
    private float solde;

    public Compte(String numero, float solde) {
        this.numero = numero;
        this.solde = solde;
    }

    public void verser(float mt) {
        solde += mt;
    }

    public void retirer(float mt) throws Exception {
        if (solde < mt) {
            throw new Exception("Solde insuffisant!");
        } else {
            solde -= mt;
        }
    }

    public float getSolde() {
        return solde;
    }

    @Override
    public String toString() {
        return "[" + numero + ", " + solde + "]";
```

```
    }
}
```

8.3.3 Classe `Main` avec Bloc `try-catch`

```
public class Main_c {
    public static void main(String[] args) {
        Compte cp = new Compte("256B301077", 500);

        try {
            cp.retirer(600); // Tentative de retrait échouée
            System.out.println("Mise à jour du solde : " + cp.
   getSolde());
        } catch (Exception e) {
            System.out.println("Erreur : " + e.getMessage());
        }
    }
}
```

Ce code empêche les actions suivantes si le retrait échoue, en capturant l'exception levée dans la méthode `retirer()`.

8.4 Gestion des exceptions avec un exemple de compte bancaire

Imaginons une situation où une partie du programme doit mettre à jour le solde du compte après chaque retrait, et que certaines actions dépendent du retrait réussi. Voici un exemple concret montrant le problème qui peut survenir sans gestion d'exceptions, suivi de la manière dont cela peut être amélioré avec gestion d'exceptions.

8.4.1 Exemple sans gestion d'exceptions

```
public class Compte {
    private String numero;
    private float solde;

    public Compte(String numero, float solde) {
        this.numero = numero;
        this.solde = solde;
    }

    public void verser(float mt) {
        solde += mt;
    }

    public void retirer(float mt) {
```

```
15          if (solde > mt) {
16              solde -= mt;
17          } else {
18              System.out.println("Solde insuffisant!");
19          }
20      }
21
22      public float getSolde() {
23          return solde;
24      }
25  }
26
27  public class Main_c {
28      public static void main(String[] args) {
29          Compte cp = new Compte("256B301077", 500);
30          cp.retirer(600);  // Tentative de retrait d'un montant
        supérieur au solde
31
32          // Autre action qui dépend de la mise à jour du solde
33          System.out.println("Mise à jour du solde : " + cp.
       getSolde());
34          System.out.println("Envoyer un mail de confirmation de
        retrait.");
35      }
36  }
```

Explication : Dans cet exemple, nous avons un compte avec un solde initial de 500. Nous tentons de retirer 600, ce qui est supérieur au solde. Le programme affiche `"Solde insuffisant!"`, mais continue d'exécuter le code après la tentative de retrait.

Le programme appelle la méthode `getSolde()`, puis imprime un message disant qu'un mail de confirmation du retrait sera envoyé.

Problème sans gestion d'exceptions : Même si le retrait échoue (solde insuffisant), le programme continue à exécuter les lignes suivantes. Cela pourrait entraîner des actions incorrectes comme :

- Envoyer un mail confirmant un retrait qui n'a pas eu lieu.
- Afficher une mise à jour du solde alors que l'opération n'a pas été effectuée.

Voici la sortie potentielle :

```
Solde insuffisant!
Mise à jour du solde : 500.0
Envoyer un mail de confirmation de retrait.
```

8.4.2 Exemple avec gestion des exceptions

Nous allons maintenant modifier cet exemple pour lever une exception lorsque le solde est insuffisant, empêchant ainsi la poursuite des actions dépendantes du retrait.

```java
public class Compte {
    private String numero;
    private float solde;

    public Compte(String numero, float solde) {
        this.numero = numero;
        this.solde = solde;
    }

    public void verser(float mt) {
        solde += mt;
    }

    public void retirer(float mt) throws Exception {
        if (solde < mt) {
            throw new Exception("Solde insuffisant!");
        } else {
            solde -= mt;
            System.out.println("Retrait effectué.");
        }
    }

    public float getSolde() {
        return solde;
    }
}

public class Main_c {
    public static void main(String[] args) {
        Compte cp = new Compte("256B301077", 500);

        try {
            cp.retirer(600);  // Tentative de retrait avec gestion d'exceptions
            // Autres actions si le retrait a réussi
            System.out.println("Mise à jour du solde : " + cp.getSolde());
            System.out.println("Envoyer un mail de confirmation de retrait.");
        } catch (Exception e) {
            // Gestion de l'erreur si le retrait échoue
            System.out.println("Erreur : " + e.getMessage());
        }
    }
}
```

Explication : Cette fois, dans la méthode `retirer()`, si le solde est insuffisant, une exception est levée avec le message `"Solde insuffisant!"`.

Dans la méthode `main()`, le bloc `try-catch` capture cette exception. Si l'exception est levée, les lignes de code après le retrait ne sont pas exécutées, et le message

d'erreur est affiché à la place.

Ainsi, la mise à jour du solde et l'envoi du mail de confirmation ne se produiront que si le retrait réussit.

8.4.3 Résultat avec gestion d'exceptions

Si le retrait échoue, voici ce qui sera affiché :

```
Erreur : Solde insuffisant!
```

Dans ce cas, aucune mise à jour du solde ni envoi de mail de confirmation n'a lieu, car le programme a arrêté l'exécution des lignes dépendantes après avoir détecté l'erreur.

8.4.4 Intérêt de l'ajout d'exceptions dans ce contexte

- **Prévention d'actions incorrectes** : Si une opération échoue (comme un retrait), vous voulez arrêter immédiatement les actions suivantes qui dépendent de cette opération.
- **Meilleure gestion des erreurs** : Le programme peut capturer et traiter les erreurs de manière contrôlée.
- **Robustesse accrue** : Les exceptions permettent au programme de ne pas continuer dans un état incohérent, évitant ainsi des résultats inattendus ou incorrects.

L'exemple avec gestion des exceptions montre clairement qu'on peut éviter les actions non souhaitées après un échec et signaler les erreurs pour qu'elles soient gérées de manière appropriée.

8.5 Exceptions Vérifiées et Non Vérifiées

8.5.1 Exceptions Vérifiées

Les exceptions vérifiées (checked) doivent être soit capturées avec un bloc `try-catch`, soit déclarées avec le mot-clé `throws`. Elles représentent des situations que le programmeur peut anticiper.

```
1 public void ouvrirFichier(String fileName) throws
     FileNotFoundException {
2     FileReader file = new FileReader(fileName);
3 }
```

8.5.2 Exceptions Non Vérifiées

Les exceptions non vérifiées (unchecked) sont des sous-classes de `RuntimeException`. Elles peuvent survenir à l'exécution et ne nécessitent pas de gestion explicite.

```
public static void main(String[] args) {
    int result = 10 / 0; // ArithmeticException non vérifiée
}
```

8.5.3 Résumé de la hiérarchie des exceptions

Pour savoir si une exception est vérifiée ou non vérifiée, il suffit de connaître sa classe parente :

— Si elle hérite de `RuntimeException` : elle est non vérifiée (unchecked).
— Si elle hérite de `Exception` mais pas de `RuntimeException` : elle est vérifiée (checked).
— Si elle hérite de `Error` (comme `OutOfMemoryError`) : c'est aussi une exception non vérifiée, mais elle représente des erreurs graves du système.

8.5.4 Exceptions de la classe `Error`

Les exceptions qui dérivent de la classe `Error` représentent des erreurs graves au niveau de la JVM. Elles sont aussi considérées comme non vérifiées, car elles ne doivent généralement pas être capturées ou gérées par le code utilisateur. Elles signalent des problèmes irrécupérables, comme :

— `OutOfMemoryError`
— `StackOverflowError`

8.5.5 Tableau récapitulatif

Type d'exception	Classe parente	vérifiée ?	Exemple
Exception vérifiée	`Exception`	Oui	`IOException`, `SQLException`
Exception non vérifiée	`RuntimeException`	Non	`NullPointerException`, `ArithmeticException`
Erreur non vérifiée	`Error`	Non	`OutOfMemoryError`, `StackOverflowError`

TABLE 8.1 – Récapitulatif des types d'exceptions en Java

8.5.6 Conclusion

Pour résumer, vous n'avez pas besoin de tester un programme pour savoir si une exception est vérifiée ou non. Il suffit de vérifier la hiérarchie de classes dans laquelle l'exception se trouve :

— **Vérifiée** si elle dérive de `Exception` mais pas de `RuntimeException`.
— **Non vérifiée** si elle dérive de `RuntimeException` ou `Error`.

8.6 Personnalisation des Exceptions

8.6.1 Créer une Exception Personnalisée

Nous allons créer une exception personnalisée appelée `SoldeInsuffisantException`.

```
public class SoldeInsuffisantException extends Exception {
    public SoldeInsuffisantException(String message) {
        super(message);
    }
}
```

8.6.2 Utilisation de l'Exception Personnalisée

Voici comment intégrer notre exception dans la gestion de comptes :

```
public void retirer(float mt) throws SoldeInsuffisantException
    {
    if (solde < mt) {
        throw new SoldeInsuffisantException("Solde insuffisant
   !");
    }
    solde -= mt;
}
```

8.6.3 Gestion des Exceptions dans `Main`

```
public static void main(String[] args) {
    Compte cp = new Compte("256B301077", 500);

    try {
        cp.retirer(600);
    } catch (SoldeInsuffisantException e) {
        System.out.println("Erreur : " + e.getMessage());
    }
}
```

8.7 Utilisation du bloc `finally`

Le bloc `finally` est une partie importante de la gestion des exceptions en Java. Il est toujours exécuté, qu'une exception soit levée ou non, et peut être utilisé pour nettoyer les ressources ou exécuter du code qui doit impérativement être exécuté après un bloc `try-catch`.

8.7.1 Exemple d'utilisation de `finally`

```
public class FinallyExample {
    public static void main(String[] args) {
        try {
```

```
4               int result = 10 / 0;  // Génère une
    ArithmeticException
5           } catch (ArithmeticException e) {
6               System.out.println("Erreur : Division par zéro !")
    ;
7           } finally {
8               System.out.println("Bloc finally exécuté.");
9           }
10          System.out.println("Programme terminé.");
11      }
12  }
```

Explication :

— Le bloc `try` tente de diviser un nombre par zéro, ce qui déclenche une `ArithmeticException`.

— Le bloc `catch` capture cette exception et affiche un message d'erreur.

— Le bloc `finally` est toujours exécuté, qu'une exception soit levée ou non. Ici, il affiche le message `"Bloc finally exécuté."`.

— Après l'exécution du bloc `finally`, le programme continue et affiche `"Programme terminé."`.

8.7.2 Quand utiliser `finally` ?

Le bloc `finally` est particulièrement utile pour :

— Libérer des ressources (par exemple, fermer un fichier ou une connexion réseau).

— Exécuter des actions qui doivent se produire, que le code dans le bloc `try` réussisse ou non.

8.8 Conclusion

Les exceptions permettent de gérer les erreurs de manière contrôlée, rendant les programmes plus robustes et sécurisés. Il est important de savoir quand utiliser des exceptions vérifiées ou non vérifiées, et comment créer des exceptions personnalisées pour répondre aux besoins spécifiques d'une application.

8.9 Exercices : Gestion des Exceptions en Java

Exercice 1 : Exceptions Non Vérifiées (Unchecked Exception)

Écrivez un programme qui tente de diviser un nombre par zéro. Utilisez un bloc `try-catch` pour capturer et gérer l'exception `ArithmeticException`.

Instructions :

— Déclarez deux variables de type `int`.

— Réalisez une division où le diviseur est zéro.

— Gérez l'exception avec un bloc `try-catch` et affichez un message d'erreur clair à l'utilisateur.
— Utilisez le bloc `finally` pour afficher un message indiquant que le programme s'est terminé correctement.

Exemple de sortie attendue :

```
Erreur : Division par zéro !
Bloc finally exécuté. Le programme s'est terminé.
```

Exercice 2 : Exceptions Vérifiées (Checked Exception)

Écrivez un programme qui demande à l'utilisateur de saisir un nombre et utilise `Integer.parseInt()` pour convertir cette chaîne en entier. Utilisez un bloc `try-catch` pour gérer l'exception `NumberFormatException` si la saisie n'est pas un nombre valide.

Instructions :

— Créez une méthode `convertirEnEntier(String nombre)` qui utilise `Integer.parseInt` pour convertir une chaîne en entier.
— Si la conversion échoue, capturez l'exception `NumberFormatException` et affichez un message d'erreur indiquant que la saisie n'est pas un nombre valide.
— Utilisez un bloc `finally` pour afficher un message indiquant que la tentative de conversion est terminée, qu'elle soit réussie ou non.

Exemple de sortie attendue :

```
Erreur : La saisie 'abc' n'est pas un nombre valide.
Bloc finally : Conversion terminée.
```

Exercice 3 : Gestion de Compte avec Exceptions

Reprenez la classe `Compte` et améliorez-la en utilisant des exceptions. Si le montant à retirer est supérieur au solde, une exception personnalisée `SoldeInsuffisantException` doit être levée.

Instructions :

— Créez une exception personnalisée `SoldeInsuffisantException`.
— Modifiez la méthode `retirer()` pour lever cette exception si le solde est insuffisant.
— Dans la méthode `main()`, capturez cette exception dans un bloc `try-catch`.

Exemple de sortie attendue :

```
Erreur : Solde insuffisant !
```

Exercice 4 : Saisie utilisateur et `InputMismatchException`

Écrivez un programme qui demande à l'utilisateur d'entrer un nombre à virgule flottante. Si l'utilisateur entre une valeur incorrecte (par exemple, du texte), le programme doit gérer l'exception `InputMismatchException`.

Instructions :

— Utilisez la classe `Scanner` pour lire un `float` saisi par l'utilisateur.
— Gérez l'exception `InputMismatchException` avec un bloc `try-catch`.
— Affichez un message d'erreur si l'utilisateur n'entre pas un nombre valide.

Exemple de sortie attendue :

```
Entrez un nombre : azerty
Erreur : Vous devez entrer un nombre valide.
```

Exercice 5 : Création d'une Exception pour Montants Négatifs

Écrivez un programme qui lève une exception personnalisée `MontantNegatifException` si un montant négatif est saisi lors d'un dépôt ou d'un retrait d'argent dans un compte bancaire.

Instructions :

— Créez une exception personnalisée `MontantNegatifException`.
— Modifiez les méthodes `verser()` et `retirer()` pour lever cette exception si le montant est négatif.
— Capturez cette exception dans un bloc `try-catch` dans la méthode `main()`.

Exemple de sortie attendue :

```
Erreur : Le montant à retirer ne peut pas être négatif.
```

Exercice 6 : Combinaison de plusieurs exceptions

Dans un programme de gestion d'un compte bancaire, combinez plusieurs exceptions : `SoldeInsuffisantException`, `MontantNegatifException`, et `InputMismatchExc`

Instructions :

— Modifiez la classe `Compte` pour gérer à la fois les erreurs de solde insuffisant et de montant négatif.
— Utilisez `Scanner` pour demander à l'utilisateur un montant, et gérez les exceptions `InputMismatchException`.
— Capturez toutes les exceptions dans la méthode `main()` avec des blocs `try-catch` distincts pour chaque type d'exception.

Exemple de sortie attendue :

```
Entrez un montant à retirer : -100
Erreur : Le montant à retirer ne peut pas être négatif.

Entrez un montant à retirer : 600
Erreur : Solde insuffisant !
```