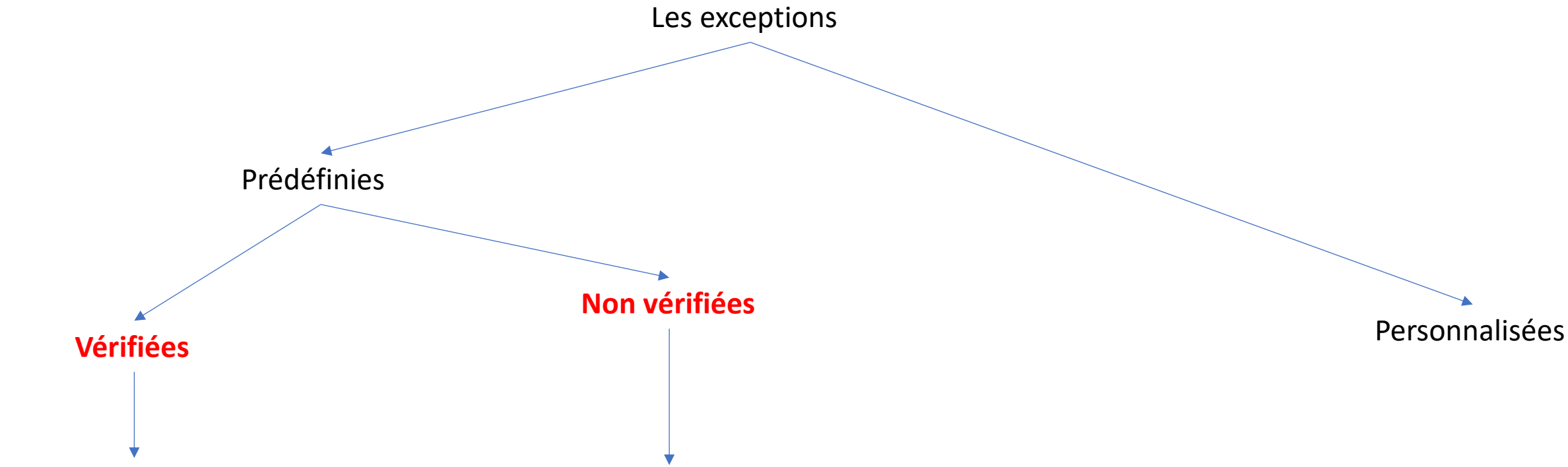
Gestion des exceptions

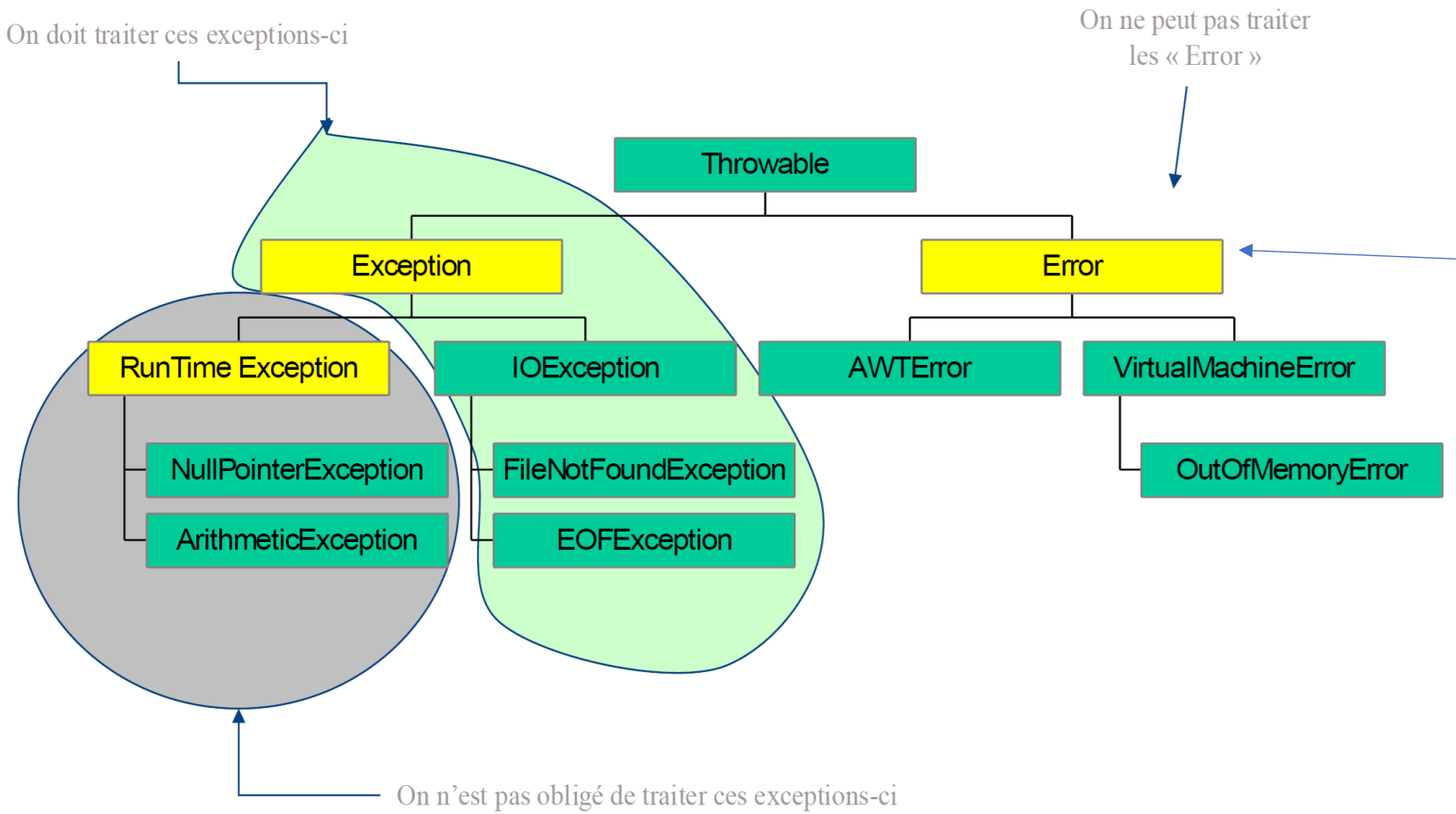
- Une **exception** est un événement imprévu qui survient pendant l’exécution d’un programme, interrompant son flux normal.

- Une exception permet de capturer et de gérer les erreurs qui pourraient autrement arrêter l'exécution du programme.

- Une exception assure que le programme peut (1) soit s’occuper d'une erreur, (2) soit s'arrêter de manière contrôlée.

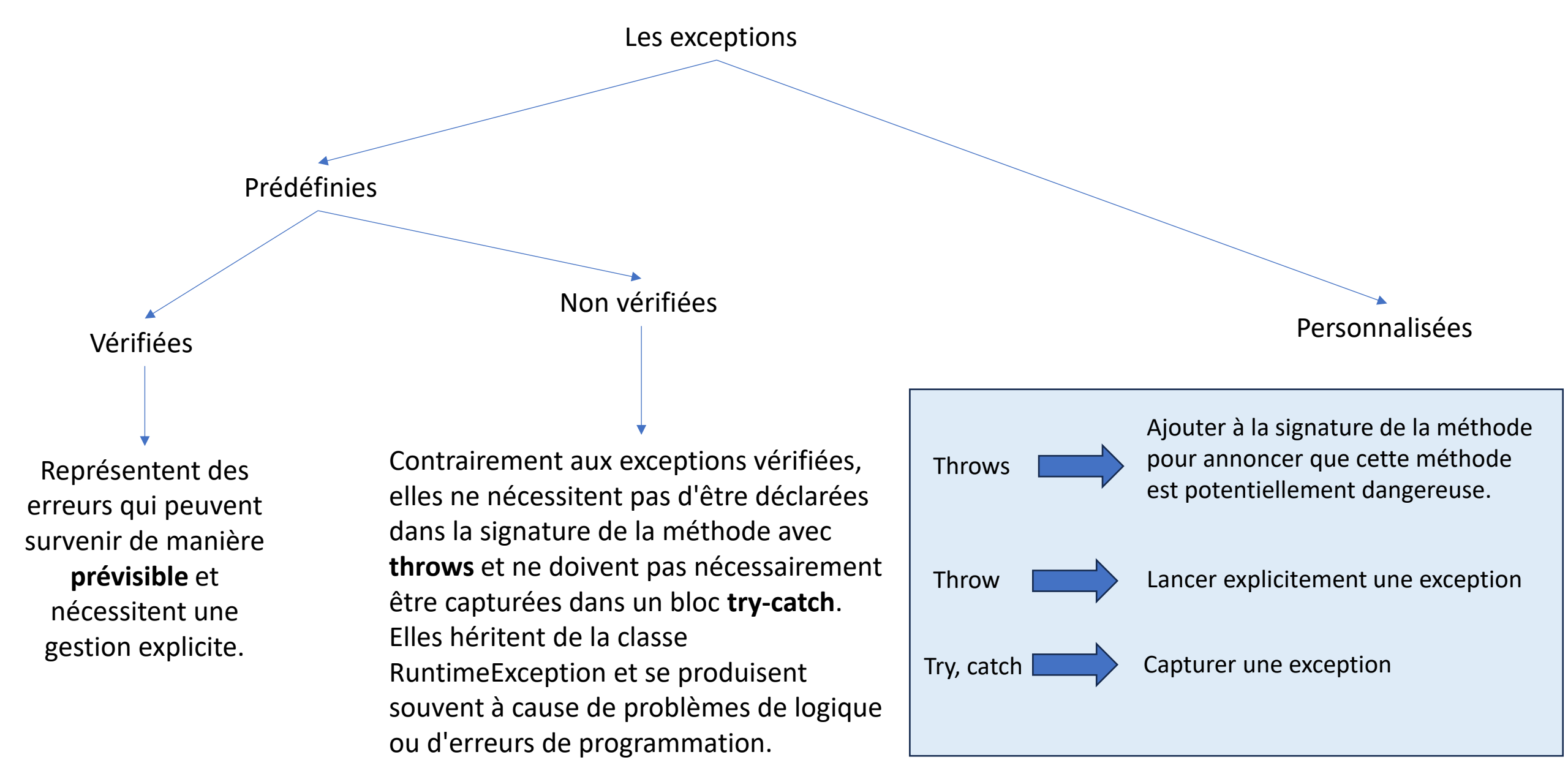


En gérant les exceptions, le développeur anticipe les erreurs et évite l’arrêt du code.



En Java, les erreurs représentent des problèmes sérieux qui surviennent dans l'environnement d'exécution de l'application et qui ne peuvent généralement pas être gérés ou récupérés par le code de l'application.

```
public class ExempleOutOfMemoryError
{
   public static void main(String[] args) {
      List<int[]> liste = new ArrayList<>();
      while (true) {
         liste.add(new int[1000000]); //
               Essai d'allouer beaucoup trop
               de mémoire
      }
   }
}
```



En gérant les exceptions, le développeur anticipe les erreurs et évite l'arrêt du code.

Quand est-ce qu’on utilise le bloc try-catch?

Quand est-ce qu’on utilise throws?

Quand est-ce qu’on utilise throw?

Bloc try-catch : Structure de Base pour la Gestion des Exceptions

Gestion des exceptions dans le code même où elles peuvent se produire.

Il capture l'exception, permettant au programme de continuer à s'exécuter sans planter, et permet au développeur de définir la façon dont l'erreur doit être traitée (capturée et gérée localement).

Ainsi, c’est une :

- Gestion immédiate dans la méthode où le problème se produit.
- Arrêt de propagation de l’exception
- Définition du comportement adapté en cas de problème : afficher des messages d'erreur, effectuer des nettoyages ou enregistrer des journaux d'erreurs, etc.

Le bloc finally :

- Exécuté après le bloc try et catch, que l'exception ait été lancée ou non.
- Son utilité paraît surtout lorsqu’on veut libérer des ressources quelque soit la finalité de notre code.

Quand est-ce qu’on utilise le bloc try-catch?

Quand est-ce qu’on utilise throws?

Quand est-ce qu’on utilise throw?

Il est possible d'utiliser plusieurs blocs catch pour gérer différents types d'exceptions dans un même bloc try.

```
try {
	int[] numbers = {1, 2, 3};
	System.out.println(numbers[5]);
		//ArrayIndexOutOfBoundsException
}
catch (ArithmeticException e) {
		System.out.println("Erreur arithmétique.");
}
catch (ArrayIndexOutOfBoundsException e) {
		System.out.println("Erreur d'indice de tableau.");
}
```

L’ordre des blocs catch est important, il faut toujours capturer les exceptions plus spécifiques avant les exceptions plus générales pour garantir que chaque exception soit traitée correctement.

Quand est-ce qu’on utilise le bloc try-catch?

Quand est-ce qu’on utilise throws?

Quand est-ce qu’on utilise throw?

Il est possible d'utiliser les méthodes prédéfinies des exceptions pour afficher les erreurs survenues:

- **getMessage()** : Récupère un message descriptif de l’exception

```
catch (Exception e) {
   System.out.println("Erreur : " + e.getMessage());
}
```

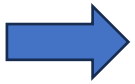
- **printStackTrace()** : Affiche le chemin de l’exception dans la pile d'appels, utile pour déboguer.

```
catch (Exception e) {
   e.printStackTrace(); // Affiche l'exception et sa trace de pile
}
```

```
Message d'erreur : / by zero
Trace de la pile :
java.lang.ArithmeticException: / by zero at
ExempleException.division(ExempleException.java:12)
at ExempleException.main(ExempleException.java:5)
```

Quand est-ce qu’on utilise le bloc try-catch?

Quand est-ce qu’on utilise throws?



Quand est-ce qu’on utilise throw?

Utilisé dans la **signature** d'une méthode pour déclarer qu'elle peut lancer une ou plusieurs exceptions vérifiées.
La méthode ne gère pas l'exception elle-même, mais la propage à la méthode appelante.
La méthode appelante devra alors gérer l'exception, soit avec un bloc try-catch, soit la propager à son tour.
Ainsi, on utilise throws lorsqu’on :

- Ne souhaite pas gérer l'exception dans la méthode où elle se produit.
- Veut permettre à la méthode appelante de décider comment gérer l'exception.
- Travaille sur des méthodes bas niveau (comme les méthodes utilitaires ou de service) qui ne devraient pas avoir à se soucier de la gestion des exceptions.

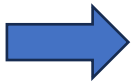
Ainsi, en propageant la gestion de l’exception, on laisse la responsabilité de la gérer à un autre niveau du programme. Cela aide à maintenir la clarté et la logique du code en déléguant la gestion des exceptions là où cela est le plus pertinent.

```
public void lireFichier(String chemin) throws
FileNotFoundException {
   FileReader lecteur = new FileReader(chemin);
   // Code de lecture du fichier
}
```

Quand est-ce qu’on utilise le bloc try-catch?

Quand est-ce qu’on utilise throws?

Quand est-ce qu’on utilise throw?



Lancer explicitement une exception, qu'elle soit prédéfinie ou personnalisée.
Contrairement à throws, qui sert à déclarer qu'une méthode peut potentiellement lancer une exception, throw est utilisé pour générer une exception à un moment précis dans le code.
Ainsi, on utilise throw pour :

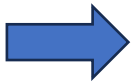
- Signaler des conditions d'erreur spécifiques : Lorsque vous souhaitez contrôler manuellement quand une exception doit être levée.
- Lancer des exceptions personnalisées : Si vous avez créé vos propres classes d'exception et que vous souhaitez les lancer dans certaines conditions.
- Forcer une exception prédéfinie : Par exemple, vous pouvez lancer une **IllegalArgumentException** si un argument passé à une méthode est invalide.

```
public void verifierAge(int age) {
   if (age < 0) {
      throw new IllegalArgumentException("L'âge
ne peut pas être négatif.");
   }}
```

Quand est-ce qu’on utilise le bloc try-catch?

Quand est-ce qu’on utilise throws?

Quand est-ce qu’on utilise throw?



Pour utiliser throw avec une exception personnalisée, il faut d'abord définir cette exception en créant une classe qui étend **Exception** (pour une exception vérifiée) ou **RuntimeException** (pour une exception non vérifiée).

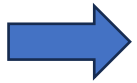
```
public class MonExceptionPersonnalisee extends Exception {
    public MonExceptionPersonnalisee(String message) {
        super(message);
    }
}
```

```
public void faireQuelqueChose(int valeur) throws
MonExceptionPersonnalisee {
    if (valeur > 100) {
        throw new MonExceptionPersonnalisee("La valeur ne
                doit pas dépasser 100.");
    }
}
```

Quand est-ce qu’on utilise le bloc try-catch?

Quand est-ce qu’on utilise throws?

Quand est-ce qu’on utilise throw?



Combinaison entre throws, throw et try-catch

```
public void verifierCondition(int valeur) throws
IllegalArgumentException {
    if (valeur < 0) {
        throw new IllegalArgumentException("La valeur ne peut
                pas être négative.");
    }
}

public void methodePrincipale() {
    try {
        verifierCondition(-5);
    } catch (IllegalArgumentException e) {
        System.out.println("Exception capturée : " +
e.getMessage());
    }
}
```

Catégorie	Exception	Description
Unchecked Exceptions	ArithmeticException	Signale une erreur arithmétique, comme une division par zéro.
	IllegalArgumentException	Indique qu'une méthode a été appelée avec un argument invalide.
	NumberFormatException	Se produit lorsqu'une chaîne est analysée comme un nombre, mais que la chaîne ne peut pas être convertie.
	NullPointerException	Se produit lorsqu'une application tente d'utiliser null à la place d'un objet.
	IndexOutOfBoundsException	Classe de base pour ArrayIndexOutOfBoundsException et StringIndexOutOfBoundsException
	StringIndexOutOfBoundsException	Indique qu'une tentative d'accès à un indice de chaîne invalide a été effectuée.
	UnsupportedOperationException	Indique qu'une opération demandée n'est pas prise en charge.
	ArrayIndexOutOfBoundsException	Indique qu'une tentative d'accès à un indice de tableau invalide a été effectuée.
	ClassCastException	Se produit lorsqu'une tentative de conversion d'un objet en un type auquel il ne peut pas appartenir.
	IllegalStateException	Indique que l'état d'une méthode est incompatible avec la méthode appelée.

```
int resultat = 10 / 0; // Déclenche ArithmeticException
```

```
String texte = null;
System.out.println(texte.length()); // Déclenche NullPointerException
```

```
int[] tableau = new int[3];
System.out.println(tableau[5]); // Déclenche ArrayIndexOutOfBoundsException
```

Catégorie	Exception	Description
Unchecked Exceptions	ArithmeticException	Signale une erreur arithmétique, comme une division par zéro.
	IllegalArgumentException	Indique qu'une méthode a été appelée avec un argument invalide.
	NumberFormatException	Se produit lorsqu'une chaîne est analysée comme un nombre, mais que la chaîne ne peut pas être convertie.
	NullPointerException	Se produit lorsqu'une application tente d'utiliser null à la place d'un objet.
	IndexOutOfBoundsException	Classe de base pour ArrayIndexOutOfBoundsException et StringIndexOutOfBoundsException
	StringIndexOutOfBoundsException	Indique qu'une tentative d'accès à un indice de chaîne invalide a été effectuée.
	UnsupportedOperationException	Indique qu'une opération demandée n'est pas prise en charge.
	ArrayIndexOutOfBoundsException	Indique qu'une tentative d'accès à un indice de tableau invalide a été effectuée.
	ClassCastException	Se produit lorsqu'une tentative de conversion d'un objet en un type auquel il ne peut pas appartenir.
	IllegalStateException	Indique que l'état d'une méthode est incompatible avec la méthode appelée.

```
try {
    int resultat = 10 / 0;
} catch (ArithmeticException e) {
    System.out.println("Erreur :
division par zéro détectée.");
}
```

```
public void setAge(int age) {
if (age < 0) {
throw new
IllegalArgumentException("L'âge
ne peut pas être négatif.");
} }
```

```
if (!connexion.estOuverte()) {
    throw new
IllegalStateException("La
connexion doit être ouverte avant
de procéder.");
}
```

Catégorie	Exception	Description
Checked Exceptions	FileNotFoundException (hérite d'IOException)	Indique que le fichier spécifié n'existe pas ou introuvable
	SQLException	Indique un problème avec une opération de base de données.
	ClassNotFoundException	Se produit lorsqu'une classe spécifique ne peut pas être trouvée au moment de l'exécution.
	InterruptedException	Indique qu'un thread a été interrompu pendant qu'il était en attente, dormant ou occupé.
	MalformedURLException	Indique une URL mal formée
	IOException	Signale une erreur d'entrée/sortie, telle que l'échec de la lecture d'un fichier.

```
public void lireFichier(String chemin) {
try {
FileReader lecteur = new FileReader(chemin);
// Code de lecture du fichier
} catch (FileNotFoundException e) {
System.out.println("Le fichier est introuvable :
" + e.getMessage()); } }
```

```
try {
      BufferedReader lecteur = new
BufferedReader(new FileReader(cheminFichier));
      String ligne;
      while ((ligne = lecteur.readLine()) != null) {
                                    System.out.println(ligne); }
      lecteur.close();
} catch (IOException e) {
          // Gestion de l'exception IOException
          System.out.println("Une erreur s'est
                produite lors de la lecture du fichier : "
                + e.getMessage());
}
```

Exercice 1 : Gestion de l'exception ArithmeticException

Écrivez une méthode **calculerDivision(int a, int b)** qui divise a par b. Si b est égal à 0, vous devez gérer l'exception **ArithmeticException** et afficher un message approprié au lieu de laisser l'exception se propager. Dans le main, appelez la méthode avec différents paramètres (y compris une division par zéro).

Exercice 2 : Gestion de l'exception IllegalArgumentException

Créez une méthode **calculerMoyenne(int[] notes)** qui calcule la moyenne des notes dans un tableau. Si le tableau est vide, la méthode doit lancer une **IllegalArgumentException**. Dans le main, appelez cette méthode et gérez l'exception. Affichez un message si l'exception est capturée.

Exercice 3 : Multicatch

Écrivez une méthode **traiterInput(String input)** qui essaie de convertir input en entier. Utilisez un bloc try avec plusieurs clauses catch pour gérer à la fois **NumberFormatException** (si input n'est pas un entier) et **NullPointerException** (si input est null). Affichez un message approprié pour chaque type d'exception.

Exercice 4 : Gestion des exceptions avec throws, throw et try-catch

Créez une classe Calculatrice contenant 4 méthodes :

- addition(int a, int b) : Renvoie la somme de a et b.
- soustraction(int a, int b) : Renvoie la différence de a et b.
- multiplication(int a, int b) : Renvoie le produit de a et b.
- division(int a, int b) : Renvoie le quotient de a par b.

Lever l'exception ArithmeticException lorsque l'utilisateur tente de diviser par zéro.

Créez une classe Main avec une méthode main pour tester votre classe Calculatrice.

Exercice 5 : Etendre notre calculatrice pour manipuler des tableaux

Modifier notre classe Calculatrice en CalculatriceAvancee et ajouter 6 nouvelles méthodes :

- **manipulationTableau:** Demandez à l'utilisateur de vous remplir un tableau d'une taille donnée. Affichez-le.
- **moyenne:** Calculez la moyenne des éléments d'un tableau.
- **maximum:** Trouvez le plus grand élément d'un tableau.
- **minimum:** Trouvez le plus petit élément d'un tableau.
- **somme:** Calculez la somme des éléments d'un tableau.
- **tri:** Triez les éléments d'un tableau.

Lever les exceptions suivantes :

- **IndexOutOfBoundsException:** Cette exception est levée lorsque vous essayez d'accéder à un élément d'un tableau en utilisant un index invalide (trop grand ou trop petit).
- **NullPointerException:** Cette exception est levée si vous essayez d'accéder à un tableau null.
- **InputMismatchException:** Cette exception est levée si vous essayez d'ajouter un élément de type incompatible dans un tableau.

Utiliser votre classe Main pour tester votre classe CalculatriceAvancee. Appelez les différentes méthodes de CalculatriceAvancee avec des valeurs valides et invalides pour vérifier la gestion des exceptions.

```
public class Exemple1 {
    public static void main(String[] args) {
        System.out.println(calculerDivision(10, 2));
                // Doit afficher 5
        System.out.println(calculerDivision(10, 0));
                // Doit afficher un message d'erreur
    }

    public static int calculerDivision(int a, int b) {
        try {
                return a / b; // Tente de diviser
        } catch (ArithmeticException e) {
            System.out.println("Erreur : Division par
                        zéro !"); // Gère l'exception
            return 0; // Retourne une valeur par
                        défaut
        }
    }
}
```

```
public class Exemple2 {
    public static void main(String[] args) {
        try {
            int[] notes = {}; // Tableau vide
            System.out.println("La moyenne est : " +
                                        calculerMoyenne(notes));
        } catch (IllegalArgumentException e) {
            System.out.println("Erreur : " + e.getMessage()); // Gère
                                                        l'exception
        }
    }

    public static double calculerMoyenne(int[] notes) {
        if (notes.length == 0) {
            throw new IllegalArgumentException("Le tableau de notes
                        ne peut pas être vide !"); // Lancer l'exception
        }
        double somme = 0;
        for (int note : notes) { somme += note; }
        return somme / notes.length; // Calculer la moyenne
    }
}
```

```
public class Exemple3 {
    public static void main(String[] args) {
        traiterInput("123"); // Doit afficher "Valeur entière : 123"
        traiterInput("abc"); // Doit afficher un message d'erreur pour NumberFormatException
        traiterInput(null);   // Doit afficher un message d'erreur pour NullPointerException
    }

    public static void traiterInput(String input) {
        try {
            int valeur = Integer.parseInt(input); // Tente de convertir la chaîne en entier
            System.out.println("Valeur entière : " + valeur); // Affiche la valeur
        } catch (NumberFormatException e) {
            System.out.println("Erreur : La chaîne n'est pas un entier valide."); // Gère l'exception pour format
        } catch (NullPointerException e) {
            System.out.println("Erreur : L'entrée est null."); // Gère l'exception pour null
        }
    }
}
```

```
public class Calculatrice {
        public int division(int a, int b) {
                if (b == 0) { throw new ArithmeticException("Division par zéro"); }
                return a / b;
        }
        // ... autres méthodes
}
public class Main {
        public static void main(String[] args) {
                Calculatrice calculatrice = new Calculatrice();
                try { int resultat = calculatrice.division(10, 0);
                        System.out.println(resultat);
                } catch (DivisionParZeroException e) {
                                System.out.println("Erreur : " + e.getMessage());
                        }
        }
}
```

```
public class CalculatriceAvancee {
        // ... méthodes existantes
        public double moyenne(int[] tableau) {
                if (tableau == null || tableau.length == 0) {
                throw new IllegalArgumentException("Tableau vide ou null");
                }
                int somme = 0;
                for (int i = 0; i < tableau.length; i++) {
                        somme += tableau[i];
                }
                return (double) somme / tableau.length;
        }
        // ... autres méthodes pour les opérations sur les tableaux
}
public static void main(String[] args) {
        CalculatriceAvancee calculatrice = new CalculatriceAvancee();
        int[] nombres = {2, 4, 6, 8};
        try {
                double moyenne = calculatrice.moyenne(nombres);
                System.out.println("La moyenne est : " + moyenne);
                // Exemple pour provoquer une IndexOutOfBoundsException
                int valeur = nombres[10];
        } catch (IllegalArgumentException e) { System.out.println("Erreur : " + e.getMessage()); }
        catch (IndexOutOfBoundsException e) { System.out.println("Index hors limites : " + e.getMessage()); } }
```

Exemple d’une exception vérifiée : **ClassNotFoundException**

```
public class ExempleExceptionVerifiee {

    public static void main(String[] args) {
        try {
            // Essayer de charger une classe avec un nom incorrect
            Class<?> maClasse = Class.forName("com.example.MaClasseInexistante");
            System.out.println("Classe trouvée : " + maClasse.getName());
        } catch (ClassNotFoundException e) {
            // Gérer l'exception vérifiée en l'attrapant
            System.out.println("Erreur : La classe spécifiée n'a pas été trouvée.");
            e.printStackTrace();
        }
    }
}
```