Conception et Programmation Objet - GM3

Exceptions et la valeur null

Mathieu Bourgais

2023

Plan

Principe des exceptions

- Lors de la création d'un programme informatique, il y a des opérations dont on sait à l'avance qu'elle peuvent créer des erreurs.
- On a alors envie, en tant que développeur, de traiter ces erreurs à l'avance pour ne pas faire planter le programme.
- En plus des erreurs classiques, on peut aussi avoir envie de créer son propre jeu d'erreurs qui seront liées aux nouvelles classes et méthodes créées.
- Le but global est de rendre son code plus propre, aussi bien pour de futurs développeurs mais aussi pour le client.

Traitement des erreurs

- On pourrait traiter ces erreurs directement dans le code (en renvoyant un code d'exécution comme cela peut être fait en C).
- On préférerait dé-corréler le code "effectif" du programme et la partie traitement des erreurs.
- On va alors dire que ces erreurs sont des **exceptions** qui vont pouvoir être détectées "à la volée" et traitées à part.
- Ces exceptions étant alors des **objets à part**, on va pouvoir les manipuler et les "transporter" dans la pile des appels.

Les exceptions en Java

- Java propose de gérer ces erreurs par le biais de la classe **Exception**
- La classe *Exception* hérite de la classe **Throwable**
- La logique de fonctionnement est la suivante :
  - Lorsqu'une erreur apparaît, une exception (héritant de *Exception*) est créée
  - Cette exception est alors envoyée dans la pile des appels
  - Il est alors possible "d'attraper" cette exception et de réaliser un traitement
- La classe *Throwable* propose des méthodes comme **printStackTrace()** ou **toString()** qui peuvent être utiles pour comprendre les erreurs complexes

Attraper et traiter une exception

- Imaginons une méthode *methode1()* dont on a repéré qu'elle pouvait envoyer une exception de type TypeException (par exemple une division par zéro qui est une *ArithmeticException*)
- On indique d'abord au programme de "surveiller" l'exécution de cette méthode en la plaçant dans un bloc **try {...}**
- On indique ensuite quoi faire si une exception est levée avec un bloc **catch (TypeException e){...}**
- On peut enfin renseigner un bloc **finally{...}** pour indiquer ce que le programme doit faire avant de s'arrêter (par exemple fermer des fichier ou des scanners).

Exemple générique d'un try/catch

```
try{
    maMethode(); //je sais que ma methode peut
        envoyer des erreurs de type TypeException1
        et TypeException2
}
catch(TypeException1 e){
    //on execute cette partie si une exception de
        type TypeException1 est levee
    System.out.println(e.toString());
}
catch(TypeException2 e){
    System.out.println(e.printStackTrace());
}
finally {
    //Partie optionnelle executee meme en cas d'
        erreur
}
```

Propager des exceptions

- On peut savoir qu'une méthode va créer une exception et pour autant préférer que cette exception soit traitée "plus haut" (par une méthode appelante)
- On va alors propager l'exception dans la pile des appels, possiblement jusqu'au *main*
- Si l'erreur arrive non-traitée au *main*, c'est là que le programme plante
- Pour indiquer qu'une méthode propage une exception, on utilise le mot-clé **throws** dans la signature de la méthode.

Exemple de propagation d'exception

```
methode1() throws TypeException{
    //code de la methode qui va creer une
        TypeException
}
methode2() throws TypeException{
    methode1();
    //puisque methode2() ne traite pas l'exception
        , celle-ci est encore propagee
}
methode3(){
    try{
        methode2();
    }
    catch(TypeException e){
        System.out.printl(e.toString();)
    }
}
```

Lever une exception

- S'il y a des cas qui vont "naturellement" lever des exceptions (parce que déjà implémentés dans des super-classes), on peut aussi souhaiter soi-même lever sa propre exception
- Pour cela, lorsqu'on tombe sur une situation qui "doit" lever une exception, on l'indique avec le mot-clé **throw**
- On va alors créer une nouvelle instance d'une classe héritant d'*Exception* qui sera propagée, information aussi indiquée par le *throws* dans la signature de la méthode

Exemple de la levée d'exception

```
private int[] tableauEntierPositifs;

public void ajouterEntierPositif(int i, int pos)
    throws IllegalArgumentException {
     if(i < 0){
          throw new IllegalArgumentException("i<0");
     } else {
          tableauEntierPositif[pos]=i;
     }
}
```

On pourra ensuite attraper cette exception dans une autre méthode.

Plan

Créer ses exceptions

- Il existe de base, en Java, de très nombreuses exceptions (**ArithmeticException**, **NullPointerExeption**, **IndexOutOfBoundException**, **FileNotFoundException**, etc.)
- Mais il est possible qu'il y ait des comportements que vous considérez comme des erreurs en fonction de votre programme (un personnage sans vie qui essaye d'agir par exemple)
- Par défaut, Java n'a aucun moyen de comprendre que ces comportements sont des erreurs du point de vue du développeur
- Il faut donc créer son propre jeu d'exceptions (que vous traiterez ensuite comme nous venons de le voir)

Exemple de création d'une exception

On doit définir une classe qui étend **Exception** et qui doit contenir au moins un constructeur (on aime bien en mettre 2, un constructeur vide et un constructeur avec une chaîne de caractère).

```
public class MonException extends Exception{
    public MonException(){
        super();
    }

    public MonException(String s){
        super(s);
    }
}
```

Aller plus loin

- On peut tout à fait créer un diagramme de classe d'exceptions (des exceptions qui héritent d'autres exceptions que l'on a créées). Cela peut permettre de mieux spécifier une erreur.
- On peut créer un chaînage d'exception i.e. lorsqu'on attrape une exception, on en lance une autre plus précise qui sera traitée par une autre méthode.
- Soyez cohérents avec l'utilisation des exceptions pour les traiter au bon endroit et ne pas surcharger vos méthodes avec plus de 3 exceptions propagées.

Plan

Utilisation de null

- On peut avoir besoin, ponctuellement, de déclarer une variable complexe vers une valeur "vide"
- On peut alors lui affecter la constante **null**
- Cette valeur *null* est équivalente à un pointeur vers un espace vide
- Attention, cela peut mener à des exceptions (*NullPointerException* par exemple)
- Une bonne pratique est de mettre des *null-check*, notamment dans la méthode *equals()*

Exemples de null-check

```
public boolean equals(Object o){
    if(o == null){
        return false;
    }
}

public void maMethode(Personnage p){
    if (p != null){
        //traitements
    }
}
```

Plan

Modifications

- Ajout des null-check dans les *equals()*
- Gestion d'une *NullPointerException* si on rentre un nom de salle qui n'est pas renseigné
- Ajout d'une exception *PersonnageInactifException* lorsqu'un personnage dont la vie est à 0 essaye de parler

Exception de personnage inactif

```
public class PersonnageInactifException extends
    Exception{

    public PersonnageInactifException() {
        super();
    }

    public PersonnageInactifException(String message)
        {
        super(message);
    }
}
```

Modification du main

```
Humain bob = null;
if(dudley.equals(bob)){
    System.out.println("Probleme␣!!!!");
} else {
    System.out.println("Tout␣va␣bien␣!");
}

try{
    hermione.parler();
}
catch (PersonnageInactifException e){
    System.out.println("ce␣personnage␣ne␣peut␣pas␣
        parler␣car␣sa␣vie␣est␣a␣0");
}
```

Gestion du mauvais nom de salle

```
String couleur = scan.next();
Personnage persoTemp = null;
try{
    persoTemp = monDonjon.getMesSalles().get(couleur).
        getPerso();
}
catch (NullPointerException e){
    System.out.println("Mauvaise salle, vous allez en
        salle verte");
    persoTemp = monDonjon.getMesSalles().get("vert").
        getPerso();
}
System.out.println(hermione + " fait face a " +
    persoTemp);
```

Résultat de l'exécution

```
Tout va bien !
ce personnage ne peut pas parler car sa vie est à 0
Quelle salle voulez-vous explorer ?
jaune
Mauvaise salle, vous allez en salle verte
nom=Hermione Granger fait face à nom=Docteur Quinn
```

Retrouvez l'ensemble du code sur Moodle.