TELECOM SudParis

IP PARIS

CSC4102 : Programmation orientée objet en JAVA — exceptions et tests

Denis Conan, avec Christian Bac et Paul Gibson

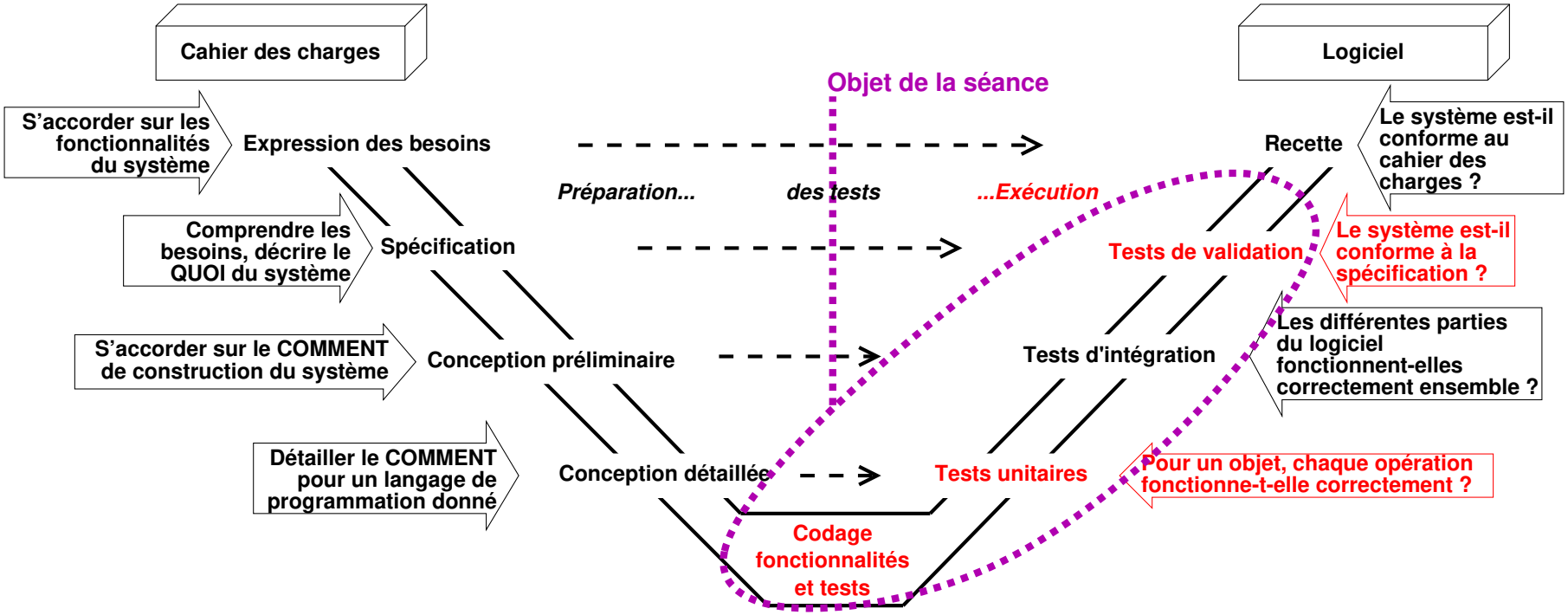
Janvier 2024

Sommaire

1 Motivations et objectifs de la séance

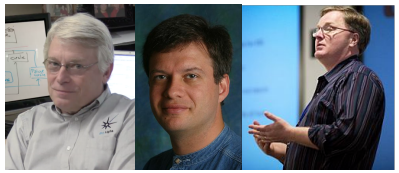
1.1 Objet de la séance

- Programmation de l'application avec quelques tests
  - Gestion des cas d'erreur avec les exceptions
  - Programmation des tests unitaires ainsi que des tests de validation



1.2 Pourquoi des exceptions et des assertions ?

R.C. Martin, « *Clean Code : A Handbook of Agile Software Craftsmanship* » [Martin, 2009]
A. Hunt and D. Thomas, « *The Pragmatic Programmer* » *[Hunt and Thomas, 2000]*
« Manifeste pour le développement agile de logiciels » [Beck et al., 2001]



- *Tip 33 : If it can't happen, use assertions*
  - *Testing doesn't find all the bugs*
  - *Your first line of defense is checking for any possible error, and your second is using assertions [exercising invariants] to try to detect those you've missed*
- *Tip 34 : Use exceptions for exceptional problems*
  - *Error handling is important, but if it obscures logic, it's wrong*
    - *The problem is that the caller must check errors immediately after the call*
    - *Unfortunately, it's easy to forget*
    - *[This is so because] two concerns are tangled : the algorithm and error handling*
  - *Use exceptions rather than return codes*
    - Exceptions allows for separating those concerns

1.2.1 Pratiquement dans ce module

Un exemple parmi les exemples représentatifs

- De (ce qui a pu être écrit dans la conception détaillée)

```
void emprunter()
    si ¬ (empruntable ∧ ¬emprunté) alors levée d'une exception
    emprunte = true
    genre.emprunter()
    nbEmprunts++
    assert invariant()
```

- À (ce qui est programmé)

Class `seance6.exception.mediathequesimplifiee.Document`

```
1   // possible levee exception sous controle : declar. levee avec throws
2   public void emprunter() {
3       // levee avec creation d'un objet exception
4       throw new IllegalStateException("Document␣non␣empruntable" + this);
5     }
6     if (emprunte) {
7       throw new IllegalStateException("Deja␣Emprunte␣" + this);
8     }
9     emprunte = true;
10    genre.emprunter();
11    nbEmprunts++;
12    assert invariant(); }
```

1.3 Quels sont les objectifs de la séance ?

- Programmation des classes
- Uniquement ce qui est nécessaire pour la réalisation des premiers cas d'utilisation du logiciel
- Programmation des tests unitaires et des tests de validation
  - Tests unitaires de quelques méthodes d'une classe importante avec invariant formulé et programmé, et participant aux premiers cas d'utilisation
  - Tests de validation de ces cas d'utilisation importants

2 Retours sur les exceptions

. Dans ce module, nous ignorons l'instruction *try-with-resources* introduite en JAVA 7

2.1 Principe de fonctionnement

- Distinction de trois parties dans le code (blocs d'instructions séparés)

1. Partie de code à l'origine de la détection d'une situation anormale
   - Interruption du flot d'exécution normale
   - Levée d'une exception (objet exception : type, message, etc.)
   - Début de la remontée du chemin d'appel
2. Région gardée : partie de code « surveillant » la levée d'une exception
   - L'une des méthodes appelées a détecté une situation anormale
   - Puis
     - Soit passage à la partie traitement de la situation anormale
     - Soit continuation de la remontée d'appel
3. Partie de code spécifique pour le traitement des situations anormales
   - « Réception » d'une exception (objet exception : type, message, etc.)
   - Différents types de traitements : cf. diapositive qui suit

2.1.1 Traitement d'une exception

- Méthode appelante qui traite la situation anormale, qui a été détectée dans la méthode appelée et surveillée

1. Compensation et poursuite
   - Exemple de compensation : message à l'utilisateur final lui indiquant pourquoi cela n'a pas fonctionné, voire avec une proposition pour ré-essayer
   - Autre exemple : des données redondantes sont utilisées pour corriger les données en erreur
   - Cela dépend du cas applicatif
2. Transformation du type de l'exception et levée d'une exception d'un type plus « adéquat »
   - Par exemple, une défaillance très technique (p.ex. problème sur un *socket* réseau) est transformée en une défaillance fonctionnelle (p.ex. problème de communication rendant le service indisponible)
   - Cela demande une modélisation des types d'exception « fonctionnelle »
     - Modélisation que nous ne faisons pas dans ce module
3. Arrêt franc avec sortie prématurée (`System.exit`) : Stratégie *fail fast*
   - Avec journalisation/affichage de la trace d'exécution (`printStackTrace()`)

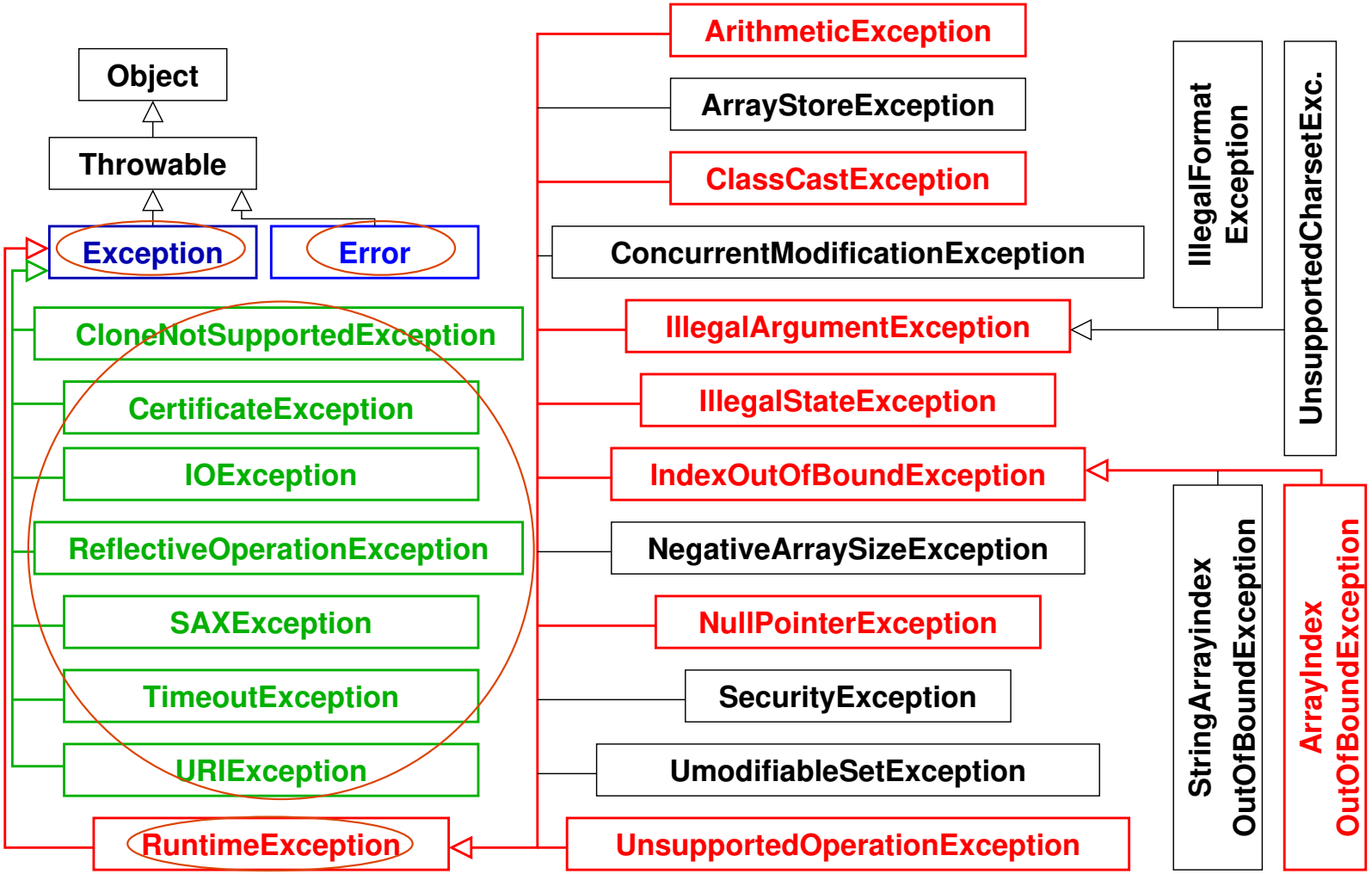
2.1.2 Mise en œuvre

Les méthodes peuvent :

- Lever une exception (dans la méthode appelée et surveillée)
  - En prévenant les autres méthodes par l'ajout d'une clause `throws` dans le prototype
    P.ex. `int maMéthode() throws MonException`
  - En utilisant le mot réservé « `throw` » pour provoquer la remontée du chemin d'appel
    + en passant en argument à `throw` un objet du type de l'exception
      P.ex. `throw new MonException("nom message")`
  - — `MonException` est une classe qui dérive de `java.lang.Exception`
- Relayer des exceptions (dans la méthode appelante)
  - En ajoutant au prototype le mot clef « `throws` » et la liste des exceptions
    P.ex. `void m() throws MonException {... int a = maMéthode(); ... }`
- Traiter des exceptions dans des blocs `try, catch` (dans la m. appelante)

2.2 Types d'exceptions et erreurs

2.2.1 Arbre d'héritage des exceptions de `java.lang`



2.2.2 Exception *versus* erreur

Selon la documentation Javadoc de Oracle

- `java.lang.Exception`
  - *Indicates conditions that a reasonable application might want to catch*
- `java.lang.Error`
  - *Indicates serious problems that a reasonable application should not try to catch*
    - P.ex. `AssertionError`, `CoderMalfunctionError`, `VirtualMachineError`
  - *Not required to [be] declare[d] in [...] `throws` clause*
    - *Regarded as unchecked exceptions for [...] compile-time checking*

2.2.3 Exception hors contrôle *vs.* sous contrôle

- Hors contrôle (en anglais, *unchecked*)
  - Sous-classes de `RuntimeException`
  - Servent à gérer les fautes de programmation
    - P.ex. pointeur `null`, division par zéro, dépassement des limites d'un tableau
    - Peuvent être évitées avec une meilleure programmation
  - Pas besoin de déclarer leur levée avec `throws` dans les prototypes
- Sous contrôle (en anglais, *checked*)
  - Autres exceptions (applicatives) déclenchées par vos méthodes
  - Vous devez les signaler aux méthodes appelantes par la directive `throws`

2.3 Exemple de classe exception (fonctionnelle)

- *extends* une classe de la hiérarchie des exceptions
  - Classe enfant directe de `Exception` $\Longrightarrow$ exception sous contrôle
- Constructeur avec un argument `String` pour le message

Class `seance6.exception.OperationImpossible`

```
1 package eu.telecomsudparis.csc4102.cours.seance6.exception;
2
3 public class OperationImpossible extends Exception {
4         public OperationImpossible(final String message) {
5                 super(message);
6         }
7 }
```

2.4 Exemple de levée d'exception sous contrôle (OperationImpossible)

- Déclaration de levée dans le prototype : `throws`
- Situation anormale détectée $\Longrightarrow$ levée de l'exception avec `throw new`
  - Dans le module, programmation des cas d'erreur fonctionnelle (p.ex. dans la façade) avec levée de l'exception `OperationImpossible`

Class `seance6.exception.mediathequesimplifiee.Mediatheque`

```
1 public boolean metEmpruntable(final String code) throws OperationImpossible {
2   Document d = chercherDocument(code);
3   if (d == null) {
4     throw new OperationImpossible("MetEmpruntable code inexistant:" + code);
5   } else {
6     d.metEmpruntable();
7   }
8   return true;
9 }
```

2.5 Exemple de levée d'exception hors contrôle (IllegalArgumentException)

- Exception hors contrôle
  $\Longrightarrow$ pas besoin de déclarer (par `throws`) dans le prototype
  - Dans le module, programmation défensive hors façade avec levée de `IllegalArgumentException`

Class `seance6.exception.mediathequesimplifiee.Document`

```
1  protected Document(final String code, final Localisation localisation,
2                     final String titre, final String auteur,
3                     final String annee, final Genre genre) {
4    if (code == null || code.equals("")) {
5          // levee exception hors controle => pas besoin de declaration throws
6      throw new IllegalArgumentException("Mauvais code (" + code + ")"); }
7    // utilisation des methodes de la classe Objects
8    Objects.requireNonNull(localisation, "Localisation ne peut pas etre null");
9    //...
10   this.code = code; this.localisation = localisation;   this.titre = titre;
11   this.auteur = auteur; this.annee = annee; this.genre = genre;
12   this.empruntable = false; this.emprunte = false; nbEmprunts = 0; }
```

2.6 Exemple de traitement avec compensation pour poursuite

- Région gardée dans le bloc `try` : là où l'on « surveille » la levée d'exception
- Traitement dans le bloc `catch`
  - Dans le module[1], compensation = affichage d'un message à la console

Class `seance6.exception.mediathequesimplifiee.InterfaceTextuelleMediatheque`

```
1  public static void main(String args[]) {
2    boolean continuer = true;
3    while (continuer) {
4      int choix = 0;
5      System.out.println("␣1-␣Ajouter␣un␣document"); //...
6      try {
7        choix = Console.readInt("\nEntrer␣le␣choix␣:");
8      } catch (Exception e) {
9        System.err.println("Erreur␣de␣saisie");
10     }
11     try {
12       switch (choix) {
13         case 1: ajouterDocument(); break; //...
14       }
15     } catch (OperationImpossible o) { // compensation = affichage
16       System.err.println("Erreur␣de␣traitement␣\"" + o.getMessage() + "\"");
17     } } }
```

1. Si on fait une interface textuelle ou un « `main` »

2.7 Exemple de traitement avec transformation du type de l'exception

- Région gardée dans le bloc `try` : là où l'on « surveille » la levée d'exception
- Traitement dans le bloc `catch`
  - Passage du type `IllegalArgumentException` au type `OperationImpossible`

Class `seance6.exception.mediathequesimplifiee.InterfaceTextuelleMediatheque`

```
1
2   static void ajouterDocument() throws OperationImpossible {
3     String code = Console.readLine("␣␣Code␣du␣document␣:");
4     String salle = Console.readLine("␣Salle␣␣:");
5     String rayon = Console.readLine("␣Rayon␣␣:");
6     String titre = Console.readLine("␣␣Titre␣:");
7     String classif = Console.readLine("␣␣Classification␣:");
8     Document doc;
9     try {
10      doc = new Audio(code, media.chercherLocalisation(salle, rayon),
11                      titre, auteur, annee, media.chercherGenre(genre),
                              classif);
12    } catch (IllegalArgumentException ex) { // transformation de type
13      throw new OperationImpossible(ex.getMessage());
14    }
```

2.8 Exemple de traitement avec plusieurs types d'exceptions

Class seance6.exception.ArrayControlled

```
1  package eu.telecomsudparis.csc4102.cours.seance6.exception;
2  public class ArrayControlled {
3    static final int max = 42;
4    public static void main(String[] args) {
5      int i, j = 0;
6      int[] arrayOfInt = new int[max];
7      for(i=0;i<arrayOfInt.length;i++){ arrayOfInt[i] = i; }
8      for(i=0;i<args.length;i++){
9        try {
10         j = Integer.parseInt(args[i]);
11         System.out.println(arrayOfInt[j]);
12       } catch (NumberFormatException nfe){
13         // premier type d'exception
14         System.out.println("Parameter␣" + args[i] + "␣should␣be␣an␣integer");
15       } catch (ArrayIndexOutOfBoundsException oobe){
16         // deuxieme type d'exception
17         System.out.println("Parameter␣" + args[i] + "␣should␣be␣>=0␣and␣<" +
               max);
18       } } } }
```

Exécution de ce programme avec les arguments « 2 43 zurg 22 » :

```
2
Parameter 43 should be >=0 and <42
Parameter zurg should be an integer
22
```

2.9 Bloc finally

- S'exécute toujours quand le bloc try se termine qu'il y ait utilisation d'un des blocs catch ou non

Class seance6.exception.BlocFinally

```
1     PrintWriter out = null;
2   try { // essai avec ouverture et ecriture ressource
3     out = new PrintWriter(new FileWriter("OutFile.txt"));
4     out.write(args[0]);
5   } catch (ArrayIndexOutOfBoundsException e) {
6     System.err.println("Caught␣ArrayIndexOutOfBoundsException:␣" + e.
          getMessage());
7   } catch (IOException e) {
8     System.err.println("Caught␣IOException:␣" + e.getMessage());
9   } finally { // bloc toujours execute
10    if (out != null) { // protection contre levee IOException
11      System.out.println("Closing␣PrintWriter");
12      out.close(); // menage ressource avant de continuer
13    } else {
14      System.out.println("PrintWriter␣not␣open"); } }
```

Exécution de ce programme avec aucun argument :

```
Caught ArrayIndexOutOfBoundsException: 0
Closing PrintWriter
```

3 Programmation/exécution des tests avec JUnit

. http://junit.org, nous utilisons JUnit 5

3.1 Classe de test

- Une classe de test (avec une série de tests) définit classiquement :
  - Des attributs utilisés par les méthodes qui suivent
  - Des méthodes de test annotées `@Test`
    - Méthode d'instance
  - Une méthode servant à l'initialisation d'un test annotée `@BeforeEach`
    - Méthode d'instance
  - Une méthode servant à la libération des données d'un test annotée `@AfterEach`
    - Méthode d'instance
  - Une méthode servant à l'initialisation de la série de tests annotée `@BeforeAll`
    - Méthode de classe
  - Une méthode servant à la libération des données de la série de tests annotée `@AfterAll`
    - Méthode de classe

3.2 Exécution d'une série de méthodes de test

- Par exemple, avec trois méthodes de test

1. Méthode de classe annotée `@BeforeAll`
2. Méthode d'instance annotée `@BeforeEach`
3. Une des méthodes[2] d'instance annotée `@Test`
4. Méthode d'instance annotée `@AfterEach`
5. Méthode d'instance annotée `@BeforeEach`
6. Une des méthodes d'instance annotée `@Test`
7. Méthode d'instance annotée `@AfterEach`
8. Méthode d'instance annotée `@BeforeEach`
9. Une des méthodes d'instance annotée `@Test`
10. Méthode d'instance annotée `@AfterEach`
11. Méthode de classe annotée `@AfterAll`

2. Aucune garantie quant à l'ordre d'exécution des méthodes de test

3.3 Méthode de test

- Une méthode de test :
  - Appelle les méthodes de l'application pour construire un scénario de test
  - Utilise les méthodes de la classe `org.junit.jupiter.api.Assertions`[3]
    - `Assertions.assertEquals`
    - `Assertions.assertTrue` et `Assertions.assertFalse`
    - `Assertions.assertNotNull` et `Assertions.assertNull`
    - `Assertions.assertSame` et `Assertions.assertNotSame`
    - `Assertions.assertArrayEquals`
    - `Assertions.fail`
    - `Assertions.assertThrows`
- Les scénarios des méthodes de test sont (par principe) indépendants
  - Les rendre dépendants est très dangereux
    - Car aucun ordre d'exécution n'est garanti

3. https://junit.org/junit5/docs/current/api/org.junit.jupiter.api/org/junit/jupiter/api/Assertions.html

3.4 Exemple de tests unitaires

Classe `seance6.mediathequesimplifie.testunitaire.TestGenre`

```
1  import org.junit.jupiter.api.AfterEach;
2  import org.junit.jupiter.api.Assertions;
3  import org.junit.jupiter.api.BeforeEach;
4  import org.junit.jupiter.api.DisplayName;
5  import org.junit.jupiter.api.Test;
6  public class TestGenre {
7      Genre g;
8      @BeforeEach // avant chaque test
9      public void setUp() {g = new Genre("Roman");}
10     @AfterEach // apres chaque test
11     public void tearDown() {g = null;}
12     @Test // un test
13     @DisplayName("constructeur␣ok") // avec un nom humain
14     public void testCtr() {
15         Assertions.assertEquals(g.getNbEmprunts(), 0);
16         Assertions.assertEquals(g.getNom(), "Roman");}
17     @Test // un autre test
18     @DisplayName("emprunts␣ok") // avec un nom intelligible
19     public void testEmprunts() {
20         for (int i = 0; i < 10; i++) {g.emprunter();}
21         Assertions.assertEquals(g.getNbEmprunts(), 10);}
22 }
```

3.5 Vérification de la levée d'une exception

Numéro de test	1	2
Nom du genre ($\neq$ `null` $\wedge \neq$ vide)	F	T
`invariant`		T
Levée d'une exception	OUI	NON
Objet créé	F	T
Nombre de jeux de test	2	1

Class `seance6.test.exception.TestGenre`

```
1  public class TestGenre {
2    @Test
3    @DisplayName("nom du genre null")
4    public void testConstructeur1Jeu1() {
5      Assertions.assertThrows(IllegalArgumentException.class, () -> new Genre(
         null)); }
6    @Test
7    @DisplayName("nom du genre chaine de caracteres vide")
8    public void testConstructeur1Jeu2() {
9      Assertions.assertThrows(IllegalArgumentException.class, () -> new Genre("
         ")); }
10   @Test
11   @DisplayName("ok")
12   public void testConstructeur2() {
13     Genre g = new Genre("un");
14     Assertions.assertTrue(g.invariant()); } }
```

3.6 Exemple de tests de validation

- Avec le patron de conception Façade, les cas d'utilisation deviennent des méthodes de la façade
- Donc, par abus de langage, on peut dire que les tests de validation sont les tests unitaires de la façade $\Longrightarrow$ on n'utilise que les méthodes de la façade

Classe `seance6.mediathequesimplifie.testdevalidation.TestEmprunter`

```
1  public class TestEmprunter {
2    private Mediatheque m1;
3    @BeforeEach public void setUp() throws Exception {
4      m1 = new Mediatheque("MediathequeTest");
5      m1.ajouterGenre("Test_genre1");
6      m1.ajouterLocalisation("Test_salle1", "Test_rayon1");
7      m1.ajouterVideo("Test_code1", "Test_salle1", "Test_rayon1", "Test_titre1"
           , "Test_auteur1", "Test_annee1", "Test_genre1", 120, "
           Test_mentionLegale1"); // ...
8    }
9    @AfterEach public void tearDown() {
10     m1 = null;
11   }
12   @Test public void emprunterDocInexistant() {
13     Assertions.assertThrows(OperationImpossible.class, () -> m1.emprunter("
           nom1", "prenom1", "Test_code4"));
14   }
```

4 Mise en pratique en TP (2h) + HP (3h)

- Étude d'exemples de gestion des exceptions pris dans le projet des exemples du module
- Continuation de la programmation de l'application avec ses tests
  - Avec programmation des exceptions en cas d'erreur, des invariants
  - Avec programmation et exécution des tests unitaires et de validation (ceux des tables de décision préparées)
- Rendu de la séance en HP : diagrammes PlantUML + `readme.md` + code
- Suivi 3 : principalement programmation, et aussi conception, spécification et préparation des tests de validation

Références I

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R., Mellor, S., Schwaber, K., Sutherland, J., and Thomas, D. (2001).
Manifeste pour le développement Agile de logiciels.

Hunt, A. and Thomas, D. (2000).
The Pragmatic Programmer : From journeyman to master.
Addison-Wesley.

Martin, R. (2009).
Clean Code : A Handbook of Agile Software Craftmanship.
Prentice Hall.