

Exceptions

Exercice 1 : Comprendre les exceptions

Dans cet exercice, on considère le programme donné au listing 1 qui compile sans erreur. Les identifiants sont volontairement non significatifs.

Listing 1 – Les exceptions en Java

```

1 class ExempleException {
2     private void m2(String p) {
3         System.out.print("<");
4         if (p == null) {
5             throw new NullPointerException();
6         }
7         if (p.length() == 0) {
8             throw new IllegalArgumentException("Chaine_vide");
9         }
10        System.out.print(p.charAt(0));
11        System.out.print(p.charAt(1));
12    }
13
14    public void m1(String p) {
15        System.out.print("[");
16        try {
17            System.out.print("]");
18        } catch (NullPointerException e) {
19            System.out.print(")");
20        }
21        catch (IllegalArgumentException e) {
22            System.out.print("N");
23        }
24        finally {
25            System.out.print("F");
26        }
27    }
28    System.out.print("]");
29}
30}
31 class ClassePrincipale {
32     public static void main(String[] args) {
33         String argument = (args.length == 0) ? null : args[0];
34         new ExempleException().m1(argument);
35     }
36 }
37 }
```

1.1 Indiquer ce qu'affiche l'exécution des commandes suivantes :

```

1 java ClassePrincipale un
2 java ClassePrincipale ""
3 java ClassePrincipale x
```

1.2 L'ordre des **catch** est-il important ?

1.3 Les exceptions `IllegalArgumentException` et `NullPointerException` sont-elles vérifiées ?

En particulier, on indiquera ce qui fait qu'une exception est vérifiée en Java, qui vérifie et l'intérêt de cette notion.

Exercice 2 : Somme des arguments de la ligne de commande

La classe Somme (listing 2) affiche la somme des nombres réels donnés en argument de la ligne de commande. Par exemple, « java Somme 10 15.5 4 » affiche 29.5.

Listing 2 – La classe Somme

```

1 /**
2  * Calculer la somme des paramètres de la ligne de commande */
3
4 /* Afficher la somme des arguments de la ligne de commande */
5
6 public static void main(String[] args) {
7     double somme = 0;
8     for (int i = 0; i < args.length; i++) {
9         somme += Double.parseDouble(args[i]);
10    }
11 }
12
13 }
```

2.1 L'exécution de java Somme 10 x 3 affiche ce qui suit dans le terminal :

```

Exception in thread "main" java.lang.NumberFormatException: For input string: "x"
at sun.misc.FloatingDecimal.readJavaFormatString(FloatingDecimal.java:2043)
at sun.misc.FloatingDecimal.parseDouble(FloatingDecimal.java:538)
at java.lang.Double.parseDouble(Double.java:538)
at Somme.main(Somme.java:8)
```

Expliquer comment interpréter cet affichage pour comprendre ce qu'il s'est passé.
2.2 Modifier la classe Somme pour afficher la somme de tous les arguments de la ligne de commande en ignorant ceux qui ne sont pas réels. On indiquera le nombre de données ignorées.

```

> java Somme 10 x 3 y
13.0
Nombre de données ignorées : 2
> java Somme 10 3
13.0
```

Exercice 3 : Livret A

Un Livret A est un produit bancaire dont le titulaire peut faire des opérations de dépôt dans la limite d'un plafond et des retraits dont le montant ne peut pas dépasser le solde du livret. Les sommes versées sur un Livret A donnent lieu à rémunération. Le taux d'intérêt ainsi que le plafond du Livret A sont identiques pour tous les Livrets A. Au 12 mars 2015, le taux d'intérêt est fixé à 1% et le plafond à 22 950 euros.

Tout comme pour les comptes courants, un historique des opérations est géré par la banque. Il est ainsi possible d'accéder aux dernières opérations de crédit ou de débit et d'édition un relevé.

- 3.1 Compléter le diagramme de classes UML du système pour faire apparaître les Livrets A.
- 3.2 Écrire la classe `LivretA`.

3.3 Proposer un programme de test de la classe `LivretA`.

Exceptions

Exercice 1: Comprendre les exceptions

1.1 /

| | | | |
|-----------------------|--------------------|-----------------------|---|
| java ClassePrincipale | un | java ClassePrincipale | " " |
| args | {"un"} length=1 | {" "} length=1 | {"x"} length=1 |
| argument | "un" | " " | "x" |
| Affichage | [(< un >)F] | [(<IF >)] | [(<x F Exception in thread "main" at java...] at ClassePrincipale.main(...) |

1.2 / Oui

1.3 / Ce sont des exceptions hors contrôle donc on ne les vérifie pas.

Exercice 2: Somme des arguments de la ligne de commande

2.1 /

2.2 / public static void main (String [] args) {

```
double somme = 0;
int NombreErreurs = 0;
try {
    for ( int i = 0 ; i < args.length ; i++ ) {
        try {
```

```
            somme += Double.parseDouble ( args [ i ] );
        }
```

```
    } catch ( NumberFormatException e ) {
```

```
        NombreErreurs++;
    }
```

```
}
```

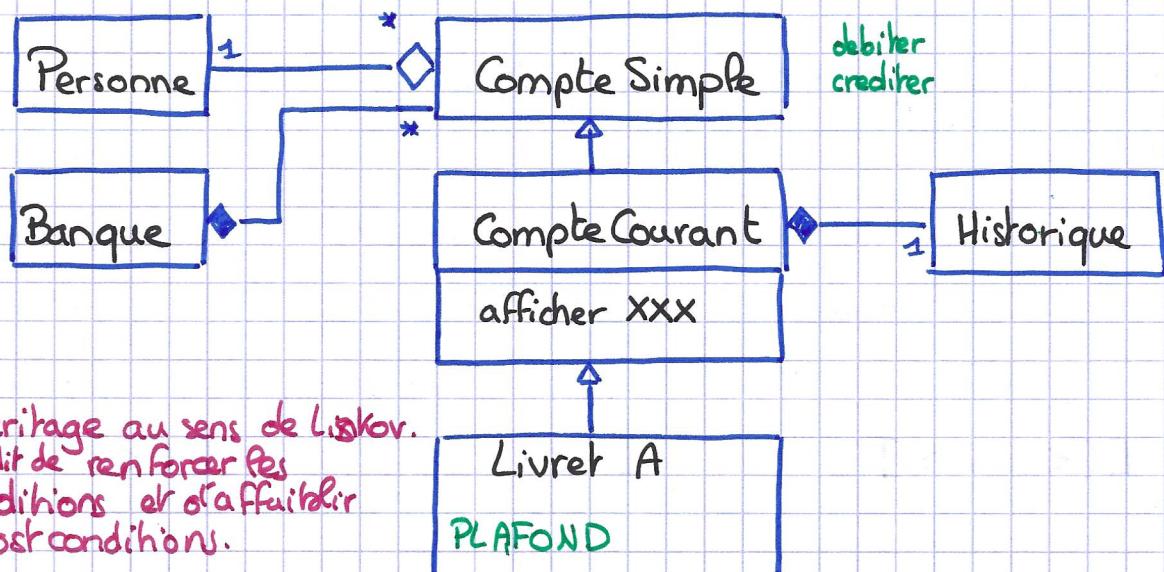
```
} finally {
```

```
    System.out.println ( somme );
}
```

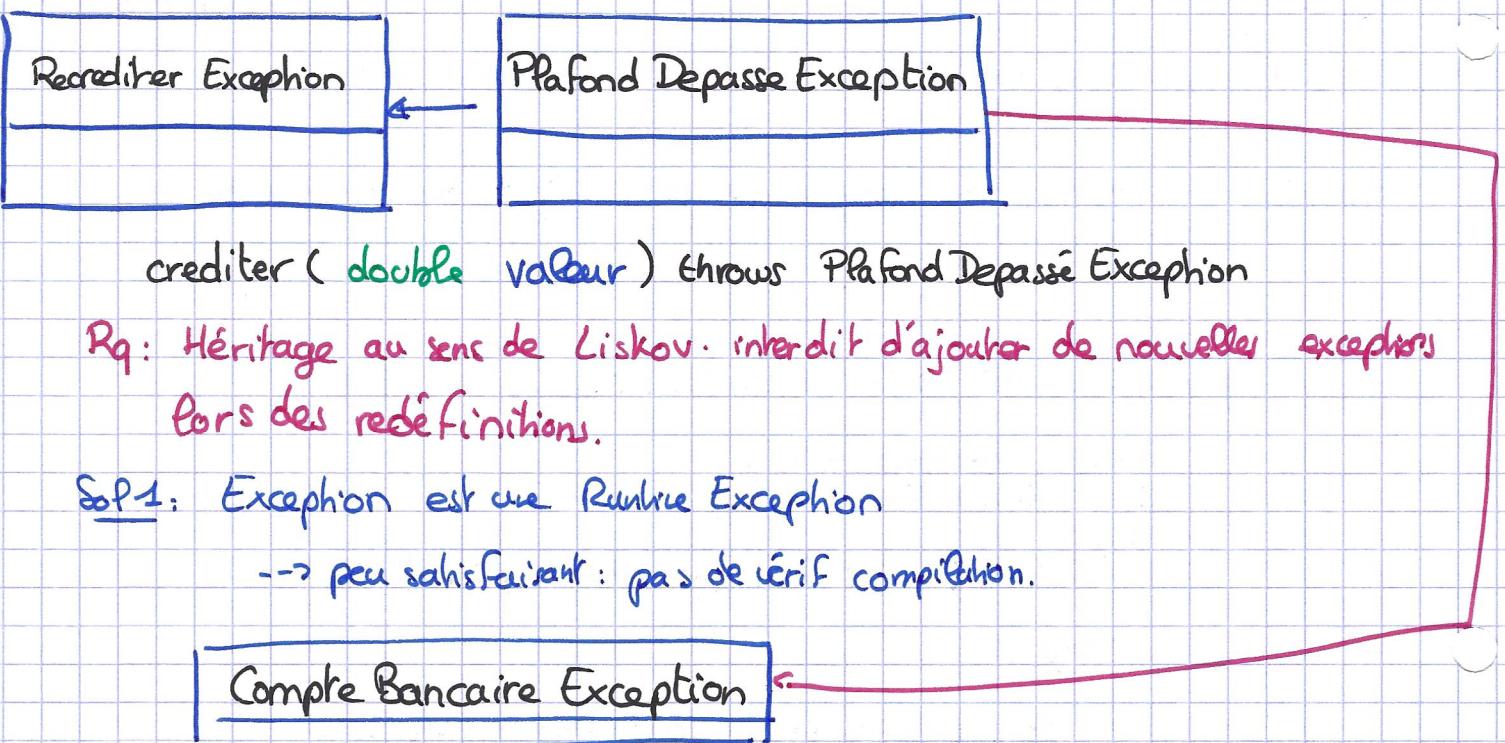
```
if ( NombreErreurs > 0 ) {
```

```
?     System.out.println ( "Nombre de données ignorées" + nombre );
```

Exercice 3 : Livret A



Rq: Héritage au sens de Liskov.
Interdit de renforcer les préconditions et d'affaiblir les postconditions.



crediter(double valeur) throws Plafond Dépassé Exception

Rq: Héritage au sens de Liskov. interdit d'ajouter de nouvelles exceptions lors des redéfinitions.

Sol 1: Exception est une RunTime Exception

--> peu satisfaisant : pas de vérif compilation.

Compte Bancaire Exception

public class LivretA extends CompteCourant.

public final static double PLAFOND = 22950 ;

public void crediter(double valeur) throws Plafond Dépassé Exception.

if (this.solde + valeur) > PLAFOND) {

throw new Plafond Dépassé Exception (

this.solde + valeur - PLAFOND.);

} else {

super.crediter(valeur)

TOB
TO 4.2

public LivretA (Personne titulaire, double solde).

throws Plafond Depasse Exception {.

if (solde > PLAFOND) {.

throw new Plafond Depasse Exception (solde PLAFOND);.

} else {.

super (titulaire, solde).

}.

}.

public class Plafond Depasse Exception extends Exception {.

public double depassent;

public Plafond Depasse Exception (double depassent) {.

this . deposit = dep

}.

}.

janv. 19, 17 22:12

CompteSimpleTest.java

Page 1/1

```
1 import org.junit.*;
2 import static org.junit.Assert.*;
3
4 /** Tests unitaires JUnit pour la classe CompteSimple.
5  * @author Xavier Crégut
6  * @version 1.4
7  */
8 public class CompteSimpleTest {
9
10    public static final double EPSILON = 1e-6;
11    // précision pour la comparaison entre réels.
12
13    protected CompteSimple c1;
14    protected CompteSimple c2;
15    protected Personne p1;
16
17    @Before
18    public void setUp() {
19        this.p1 = new Personne("Xavier", "Crégut", true);
20        this.c1 = new CompteSimple(p1, 1000);
21        this.c2 = new CompteSimple(p1);
22    }
23
24    @Test
25    public void testerInitialisationC1() {
26        assertEquals(this.c1.getTitulaire(), this.p1);
27        assertEquals(1000, this.c1.getSolde(), EPSILON);
28    }
29
30    @Test
31    public void testerInitialisationC2() {
32        assertEquals(this.c2.getTitulaire(), this.p1);
33        assertEquals(0, this.c2.getSolde(), EPSILON);
34    }
35
36    @Test
37    public void testerCrediter() {
38        this.c1.crediter(100);
39        assertEquals(1100, this.c1.getSolde(), EPSILON);
40    }
41
42    @Test
43    public void testerDebiter() {
44        this.c1.debiter(100);
45        assertEquals(900, this.c1.getSolde(), EPSILON);
46        this.c1.debiter(250);
47        assertEquals(650, this.c1.getSolde(), EPSILON);
48    }
49
50    @Test
51    public void testerSoldeNegatif() {
52        this.c1.debiter(1200);
53        assertEquals(-200, this.c1.getSolde(), EPSILON);
54        this.c1.crediter(300);
55        assertEquals(100, this.c1.getSolde(), EPSILON);
56    }
57
58    public static void main(String[] args) {
59        org.junit.runner.JUnitCore.main(CompteSimpleTest.class.getName())
60    }
61
62 }
```

⚠ Car on choisit des op sur les flottants

janv. 19, 17 22:12

CompteSimple.java

Page 1/1

```

1  /** CompteSimple modÃ©lise un compte bancaire simple tenu en euros. Il
2   * est caractÃ©risÃ© par un titulaire et un solde (positif ou nÃ©gatif)
3   * et autorise seulement les opÃ©rations de crÃ©dit et dÃ©bit.
4   * @author Xavier CrÃ©gut
5   * @version 1.9
6   */
7  public class CompteSimple {
8
9      /** Titulaire du compte. */
10     private Personne titulaire;
11
12     /** Solde du compte exprimÃ© en euros. */
13     private double solde;
14
15     /** Initialiser un compte.
16      * @param titulaire le titulaire du compte
17      * @param depotInitial le montant initial du compte
18      */
19     // {@ requires leTitulaire != null;           // le titulaire existe
20     // {@ requires depotInitial >= 0;           // montant initial strictement positif
21     // {@ ensures getSolde() == depotInitial;    // solde initialisÃ©
22     // {@ ensures getTitulaire() == leTitulaire; // titulaire initialisÃ©
23     public CompteSimple(Personne leTitulaire, double depotInitial) {
24         this.solde = depotInitial;
25         this.titulaire = leTitulaire;
26     }
27
28     /** Initialiser un compte Ã  partir de son titulaire.
29     * Son solde est nul.
30     * @param titulaire le titulaire du compte
31     */
32     // {@ requires titulaire != null;           // le titulaire existe
33     // {@ ensures getSolde() == 0;              // pas de dÃ©pÃ´t initial
34     // {@ ensures getTitulaire() == titulaire; // titulaire initialisÃ©
35     public CompteSimple(Personne titulaire) {
36         this(titulaire, 0);
37     }
38
39     /** Solde du compte exprimÃ© en euros. */
40     public /*@ pure */ double getSolde() {
41         return this.solde;
42     }
43     ↗ VÃ©rifier que ces attributs n'ont pas Ã©tÃ© modifiÃ©s par la
44     /** Titulaire du compte. */
45     public /*@ pure */ Personne getTitulaire() {
46         return this.titulaire;
47     }
48
49     /** CrÃ©diter le compte du montant (exprimÃ© en euros).
50     * @param montant montant dÃ©posÃ© sur le compte en euros
51     */
52     // {@ requires montant > 0;
53     // {@ ensures getSolde() == \old(getSolde()) + montant; // montant crÃ©ditÃ©
54     public void crediter(double montant) {
55         this.solde = this.solde + montant;
56     }
57
58     /** DÃ©biter le compte du montant (exprimÃ© en euros).
59     * @param montant montant retirÃ© du compte en euros
60     */
61     // {@ requires montant > 0;
62     // {@ ensures getSolde() == \old(getSolde()) - montant; // montant dÃ©bitÃ©
63     public void debiter(double montant) {
64         this.solde = this.solde - montant;
65     }
66
67     public String toString() {
68         return "solde=" + this.getSolde()
69             + ", titulaire=\"" + this.getTitulaire() + "\"";
70     }
71 }
72 }

Ttes ces classes on java hÃ©ritent de la classe Object
et donc de ToString()
```