Exceptions

Objectif

— Manipuler les exceptions ;

1 Entrées/sorties

Exercice 1 : Lire un entier à partir du clavier
Ajouter une méthode `readInt(String message)` dans la classe Console fournie (listing 1). Cette méthode lit, à partir du clavier, un entier. Le message passé en paramètre est affiché pour inviter l'utilisateur à taper l'entier. L'entier est lu sous la forme d'une chaîne de caractères (`String`) qui est transformée en un entier (**`int`**). Si l'utilisateur n'a pas saisi un entier, une nouvelle saisie est demandée. On utilisera les méthodes `printPrompt(String)` et `readLine()` de la classe Console (listing 1). On utilisera également la méthode `parseInt(String)` de la classe `Integer`.

Listing 1 – La classe Console

```
1  import java.io.*;
2
3  /** Quelques méthodes pour lire à partir du clavier.
4   *  @author Xavier Crégut (très largement inspiré par Cay Horstmann) */
5  public class Console {
6      /** Afficher un prompt à l'écran (sans passage à la ligne) */
7      public static void printPrompt(String prompt) {
8          System.out.print(prompt + "␣");
9          System.out.flush(); // forcer l'écriture en l'absence de '\n'
10     }
11
12    /** Lire une chaîne de caractères à partir du clavier.  La chaîne se
13      * termine par un retour à la ligne (qui n'en fait pas partie) (cf corejava)
14      * @return la ligne lue à partir du clavier (sans le retour à la ligne) */
15     public static String readLine() {
16         int ch;
17         String r = "";
18         boolean done = false;
19         while (!done) {
20             try {
21                 ch = System.in.read();
22                 if (ch < 0 || (char) ch == '\n') {
23                     done = true;
24                 } else if ((char) ch != '\r') {
25                             // weird--it used to do \r\n translation
26                     r = r + (char) ch;
27                 }
```

```
28              } catch (java.io.IOException e) {
29                  done = true;
30              }
31          }
32          return r;
33      }
34
35  }
```

2 Comptes bancaires et exceptions

Exercice 2 : Comptes simples et exceptions
L'objectif de cet exercice est d'utiliser les exceptions pour exprimer les responsabilités de la classe CompteSimple (listing 2) et de voir les conséquences sur le reste de l'application résumé ici au programme de test du listing 3.

2.1 *Les constructeurs.* Un compte simple est créé à partir d'un titulaire et d'un dépôt initial. Le titulaire doit exister (différent de **`null`**) et le dépôt initial doit être positif. On utilisera les deux exceptions TitulaireInvalideException et DépôtInitialException pour détecter ces deux cas d'erreurs.

2.1.1 Modifier le constructeur en utilisant les deux exceptions fournies qui sont non contrôlées par le compilateur Java. Quelles sont les modifications à apporter au programme de test ?

2.1.2 Modifier les deux exceptions pour qu'elles soient sous contrôle du compilateur Java. Quelles sont les modifications à apporter ?

2.2 *Les méthodes créditer et débiter.* Définir une exception MontantException qui signale que le montant passé en paramètre des méthodes créditer et débiter est incorrect.
Indication : Quelles sont les informations à associer à cette exception pour qu'elle puisse être traitée dans le gestionnaire d'exception qui décidera de la récupérer ?

2.3 *Le programme principal.* Dans le programme principal, on souhaite récupérer les exceptions sur les comptes qui se produisent sans pour autant avoir à énumérer tous les cas d'erreurs possibles. Ceci permettra en particulier de prendre en compte, sans modification, de nouveaux types d'anomalie qui pourraient être introduits par la suite sur les comptes bancaires.

2.3.1 Proposer et implanter une solution.

2.3.2 L'énoncé ci-dessus présente un avantage à la solution donnée, indiquer ses inconvénients.

Listing 2 – La classe CompteSimple

```
1  /** CompteSimple modélise un compte bancaire simple tenu en euros.  Il
2   * est caractérisé par un titulaire et un solde (positif ou négatif)
3   * et autorise seulement les opérations de crédit et débit.
4   * @author  Xavier Crégut
5   * @version 1.9
6   */
7  public class CompteSimple {
8
```

```
9       /** Titulaire du compte. */
10      private Personne titulaire;
11
12      /** Solde du compte exprimé 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      public CompteSimple(Personne leTitulaire, double depotInitial) {
20          this.solde = depotInitial;
21          this.titulaire = leTitulaire;
22      }
23
24      /** Initialiser un compte à partir de son titulaire.
25       * Son solde est nul.
26       * @param titulaire le titulaire du compte
27       */
28      public CompteSimple(Personne titulaire) {
29          this(titulaire, 0);
30      }
31
32      /** Solde du compte exprimé en euros. */
33      public double getSolde() {
34          return this.solde;
35      }
36
37      /** Titulaire du compte. */
38      public Personne getTitulaire() {
39          return this.titulaire;
40      }
41
42      /** Créditer le compte du montant (exprimé en euros).
43       * @param montant montant déposé sur le compte en euros
44       */
45      public void crediter(double montant) {
46          this.solde = this.solde + montant;
47      }
48
49      /** Débiter le compte du montant (exprimé en euros).
50       * @param montant montant retiré du compte en euros
51       */
52      public void debiter(double montant){
53          this.solde = this.solde - montant;
54      }
55
56      public String toString() {
57          return "solde=" + this.getSolde()
58              + ",␣titulaire=\"" + this.getTitulaire() + "\"";
59      }
60
61  }
```

Listing 3 – La classe TestCompteSimple1Assert

```
1  /** Programme de test sur les comptes simples (utilisant assert).
2    * @author Xavier Crégut
3    * @version    1.3
4    */
5  public class TestCompteSimple1Assert {
6      public static void main (String args []){
7          Personne p1 = new Personne("Xavier", "Crégut", true);
8          CompteSimple cs1 = new CompteSimple(p1, 1000);
9          assert cs1.getSolde() == 1000;
10         cs1.crediter(100);
11         assert cs1.getSolde() == 1100;
12         cs1.debiter(2000);
13         assert cs1.getSolde() == -900;
14
15         CompteSimple cs2 = new CompteSimple(p1);
16         assert cs2.getSolde() == 0;
17         cs2.crediter(100);
18         assert cs2.getSolde() == 100;
19     }
20 }
```

Exercice 3 : Impact des exceptions sur le compte courant

Maintenant que la classe `CompteSimple` a été modifiée pour lever des exceptions (exercice 2), nous allons en voir l'impact sur la classe `CompteCourant`.

3.1 Compiler la classe `CompteCourant`. Quelles sont les erreurs signalées ?

3.2 Corriger la classe `CompteCourant` et le programme de test `ExempleComptes`.

Exercice 4 : Livret A et exceptions

La classe `LivretA` a été écrite en faisant un héritage de la classe CompteCourant. Cet héritage a été justifié par le souhait de réutiliser la gestion du titulaire, du solde et de l'historique du CompteCourant. Ceci ne peut jamais être une justification en Java car la relation d'héritage définit une relation de sous-typage. En nous appuyant sur les exceptions, nous montrons que la relation d'héritage conduit à modifier les classes parentes (pour effectivement avoir une relation de sous-typage).

4.1 La compilation de la classe `LivretA` provoque des erreurs liées à la non déclaration des exceptions pouvant être levées par certaines méthodes. Corriger la classe `LivretA` pour supprimer ces messages.

4.2 Par rapport à un compte courant, nous avons une contrainte supplémentaire sur le montant de créditer. Il ne faut pas dépasser le plafond du Livret A.

4.2.1 Définir une exception sous contrôle, `DépassementPlafondException`. Elle sera utilisée dans la méthode `créditer` de `LivretA` pour signaler un montant qui provoquerait un dépassement du plafond.

4.2.2 Quelles sont les erreurs signalées par le compilateur lors de la compilation de `LivretA` ? Pourquoi ?

4.2.3 Que faut-il faire pour corriger les erreurs signalées ?

Remarque : Nous aurions pu mener un raisonnement analogue en nous appuyant sur les préconditions de la méthode créditer de la classe `CompteCourant` (ou `CompteSimple`) et de sa redéfinition dans `LivretA`. La précondition est en effet renforcée ce qui est contraire à la règle qui dit que la précondition d'une méthode redéfinie ne peut qu'être affaiblie.

4.3 L'utilisation de l'héritage entre `LivretA` et `CompteCourant` conduit à modifier la sémantique des classes `CompteCourant` et `CompteSimple`. Si ceci n'est pas souhaitable, il ne faut pas utiliser l'héritage.

Comment faire alors pour écrire la classe `LivretA` ?