IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

TP1 – Exécution java, gestion de la mémoire et débogage

Informatique – MAPD

Correction

Objectifs pédagogiques

- Écrire et exécuter un petit programme Java avec Eclipse
- Redéfinir une méthode en Java et utiliser l'annotation `@Override` à bon escient
- Justifier un choix de conception : mutabilité/immutabilité des nombres
- Produire une Javadoc en utilisant les *tags* de base
- Utiliser les éléments fondamentaux du débogage d'un programme Java avec Eclipse

Eclipse

Exercice 1 (*Classe Rationnel*)

Le but de cet exercice est de programmer une classe `Rational` qui permet de décrire les nombres rationnels.

Un rationnel est le quotient de deux entiers (dont le dénominateur n'est pas nul) et qui permet de calculer des sommes, des produits, l'inverse et l'opposé.

Nous allons appliquer la méthode des petits pas : on crée d'abord une version très simple puis on écrit des tests qu'on exécute, puis on complète la classe petit à petit.

▷ **Question 1.1 :**
Écrivez une version sans exception de la classe `Rational` avec un constructeur, un `main` qui instancie plusieurs rationnels (dont un avec un dénominateur nul) et les affiche (avec la méthode `toString()` de `Object`).

```
package rational;

/**
```

```
 * The Class Rational1.
 */
public class Rational1 {

  /** The numerator. */
  private int numerator;

  /**
   * Gets the numerator.
   *
   * @return the numerator
   */
  public int getNumerator() {
    return numerator;
  }

  /**
   * Gets the denominator.
   *
   * @return the denominator
   */
  public int getDenominator() {
    return denominator;
  }

  /** The denominator. */
  private int denominator;

  /**
   * Instantiates a new rational 1.
   *
   * @param n the n
   * @param d the d
   */
  public Rational1(int n, int d) {
    numerator = n;
    denominator = (d==0?1:d); // ensure @invariant non nul denominator
  }

  /**
   * The main method.
   *
   * @param args the arguments
   */
  public static void main(String[] args) {
    Rational1 r0, r1, r2, r3;
    r0 = new Rational1(10,0);
    System.out.println("r0 = " + r0);
    r1 = new Rational1(0,6);
    System.out.println("r1 = " + r1);
    r2 = new Rational1(-2,3);
```

```
    System.out.println("r2 = " + r2);
    r3 = new Rational1(1,2);
    System.out.println("r3 = " + r3);
  }
}
```

▷ **Question 1.2 :**
Redéfinissez les méthode `toString()` et `equals()`. Exécutez à nouveau.

Dans la question, il n'y a pas explicitement la signature ; c'est volontaire.
Commentez l'annotation `@Override` et le triangle vert d'Eclipse. Permet de signifier (1) l'intention de faire une redéfinition ou (2) la reconnaissance d'une redéfinition par l'IDE.
Penser à ajouter le test (o != null) avant le test de type

```
package rational;

/**
 * The Class Rational2.
 */
public class Rational2 {

  /** The numerator. */
  private int numerator;

  /** The denominator. */
  private int denominator;

  /**
   * Instantiates a new rational 2.
   *
   * @param n the n
   * @param d the d
   */
  public Rational2(int n, int d) {
    this.numerator = n;
    this.denominator = (d==0?1:d); // ensure @invariant non nul denominator
  }

  /**
   * To string.
   *
   * @return the string
   */
  @Override
  public String toString() {
    return this.numerator + (this.denominator == 1?"":"/" + this.denominator);
  }

  /**
   * Equals.
```

```
   *
   * @param o the o
   * @return true, if successful
   */
  @Override
  public boolean equals(Object o) {
    if ((o == null) || !(o instanceof Rational2)) return false;
    Rational2 r = (Rational2) o;
    return this.numerator * r.denominator == r.numerator * this.denominator;
  }

  /**
   * The main method.
   *
   * @param args the arguments
   */
  public static void main(String[] args) {
    Rational2 r0, r1, r2, r3, r4;
    r0 = new Rational2(10,0);
    System.out.println("r0 = " + r0);
    r1 = new Rational2(0,6);
    System.out.println("r1 = " + r1);
    r2 = new Rational2(-2,3);
    System.out.println("r2 = " + r2);
    r3 = new Rational2(1,2);
    System.out.println("r3 = " + r3);
    r4 = new Rational2(10,20);
    System.out.println("r4 = " + r4);
    System.out.println("r1 = r2 ? " + r1.equals(r2));
    System.out.println("r1 = r3 ? " + r1.equals(r3));
    System.out.println("r3 = r4 ? " + r3.equals(r4));
    System.out.println("r3 = null ? " + r3.equals(null));

  }
}
```

▷ **Question 1.3 :**
Ajoutez une méthode pour additionner deux rationnels. Ajoutez quelques appels de cette méthode dans le `main`. Exécutez à nouveau.

Notez qu'il n'y a qu'un argument à la méthode `sum` puisqu'on demande le calcul à un autre rationnel.
Dans la solution ci-dessous, on crée un nouveau `Rational` ; on ne modifie pas **`this`**. Les nombres sont immutables par les opérations.
La mutabilité reste interne et ne change pas la valeur de l'objet, juste sa présentation.

```
package rational;

/**
 * The Class Rational3.
```

```
 */
public class Rational3 {

  /** The numerator. */
  private int numerator;

  /** The denominator. */
  private int denominator;

  /**
   * Instantiates a new rational 3.
   *
   * @param n the n
   * @param d the d
   */
  public Rational3(int n, int d) {
    numerator = n;
    denominator = (d==0?1:d); // ensure @invariant non nul denominator
  }

  /**
   * I return a String numerator (if denominator == 1) or numerator/denominator.
   *
   * @return the string
   */
  @Override
  public String toString() {
    return numerator + (denominator == 1?"":"/" + denominator);
  }

  /**
   * Equals.
   *
   * @param o the o
   * @return true, if successful
   */
  @Override
  public boolean equals(Object o) {
    if ((o == null) || !(o instanceof Rational2)) return false;
    Rational3 r = (Rational3) o;
    return this.numerator * r.denominator == r.numerator * this.denominator;
  }

  /**
   * Sum.
   *
   * @param r the r
   * @return the rational 3
   */
  public Rational3 sum(Rational3 r) {
    return new Rational3(
```

```
        this.numerator * r.denominator + r.numerator * this.denominator,
        this.denominator * r.denominator);
  }

  /**
   * The main method.
   *
   * @param args the arguments
   */
  public static void main(String[] args) {
    Rational3 r0, r1, r2, r3, r4;
    r0 = new Rational3(10,0);
    System.out.println("r0 = " + r0);
    r1 = new Rational3(0,6);
    System.out.println("r1 = " + r1);
    r2 = new Rational3(-2,3);
    System.out.println("r2 = " + r2);
    r3 = new Rational3(1,2);
    System.out.println("r3 = " + r3);
    r4 = new Rational3(10,20);
    System.out.println("r4 = " + r4);
    System.out.println("r1 = r2 ? " + r1.equals(r2));
    System.out.println("r1 = r3 ? " + r1.equals(r3));
    System.out.println("r3 = r4 ? " + r3.equals(r4));
    System.out.println("r3 = null ? " + r3.equals(null));
    System.out.println("r1+r2 = " + (r1.sum(r2)));
  }
}
```

▷ **Question 1.4 :**
Pour réaliser l'addition, vous avez du faire un choix de conception : changer la valeur du rationnel récepteur (objet mutable) *ou* retourner un nouveau (objet non mutable). Justifiez votre choix.

Programmer, c'est faire des choix de conception qui peuvent orienter radicalement la forme du code et ses caractéristiques (telles que la réutilisabilité, la maintenabilité, l'évolutivité).

choix mutable plus simple, mais change la valeur (l'état) du nombre rationnel. 1 devient 3 par exemple. Peut-être troublant pour un nombre. (Moins pour un point qu'on translate par exemple)

choix immutable retourne un nouveau rationnel. Une fois créé un nombre ne change pas de valeur.

Les objets non mutables ont des propriétés intéressantes ; ils ne changent pas de valeur. On peut faire des calculs facilement avec. On les copie facilement.

▷ **Question 1.5 :**
Répétez la démarche précédente pour les fonctions qui permettent de multiplier, retourner l'opposé.

```java
package rational;

/**
 * The Class Rational4.
 */
public class Rational4 {

  /** The numerator. */
  private int numerator;

  /** The denominator. */
  private int denominator;

  /**
   * Instantiates a new rational 4.
   *
   * @param n the n
   * @param d the d
   */
  public Rational4(int n, int d) {
    numerator = n;
    denominator = (d==0?1:d); // ensure @invariant non nul denominator
  }

  /**
   * To string.
   *
   * @return the string
   */
  @Override
  public String toString() {
    return numerator + (denominator == 1?"":"/" + denominator);
  }

  /**
   * Equals.
   *
   * @param o the o
   * @return true, if successful
   */
  @Override
  public boolean equals(Object o) {
    if ((o == null) || !(o instanceof Rational4)) return false;
    Rational4 r = (Rational4) o;
    return this.numerator * r.denominator == r.numerator * this.denominator;
  }

  /**
   * Sum.
   *
```

```
   * @param r the r
   * @return the  rational  4
   */
  public Rational4 sum(Rational4 r) {
    return new Rational4(
        this.numerator * r.denominator + r.numerator * this.denominator,
        this.denominator * r.denominator);
  }

  /**
   * Diff.
   *
   * @param r the r
   * @return the  rational  4
   */
  public Rational4 diff(Rational4 r) {
    return this.sum(r.neg());
  }


  /**
   * Times.
   *
   * @param r the r
   * @return the  rational  4
   */
  public Rational4 times(Rational4 r) {
    return new Rational4(
        this.numerator * r.numerator,
        this.denominator * r.denominator);
  }

  /**
   * Neg.
   *
   * @return the  rational  4
   */
  public Rational4 neg() {
    return new Rational4(
        - this.numerator,
        this.denominator);
  }

  /**
   * The main method.
   *
   * @param args the arguments
   */
  public static void main(String[] args) {
    Rational4 r0, r1, r2, r3, r4;
    r0 = new Rational4(10,0);
```

```
    System.out.println("r0 = " + r0);
    r1 = new Rational4(0,6);
    System.out.println("r1 = " + r1);
    r2 = new Rational4(-2,3);
    System.out.println("r2 = " + r2);
    r3 = new Rational4(1,2);
    System.out.println("r3 = " + r3);
    r4 = new Rational4(10,20);
    System.out.println("r4 = " + r4);
    System.out.println("r1 = r2 ? " + r1.equals(r2));
    System.out.println("r1 = r3 ? " + r1.equals(r3));
    System.out.println("r3 = r4 ? " + r3.equals(r4));
    System.out.println("r3 = null ? " + r3.equals(null));
    System.out.println("r1+r2 = " + (r1.sum(r2)));
    r4 = new Rational4(-8,-4);
    System.out.println("r4 = " + r4);
    System.out.println("r1 * r2 = " + r1.times(r2));
    System.out.println("-(r3 * r4) = " + (r3.times(r4)).neg());
    System.out.println("r3 * r3 = " + (r3.times(r3)));
  }
}
```

Pour calculer le pgcd de deux entiers, on peut utiliser le code suivant :

```
private static int gcd(int a, int b) {
  for(int i = Math.min(a, b); i > 0; i--){
    if(a % i == 0 && b % i == 0) return i;
  }
  return 1;// never reach but make function compilable
}
```

▷ **Question 1.6 :**
(Avancé) Faites en sorte que les Rationnels retournés soient toujours irréductibles (rapport d'entiers premiers entre eux.).

Introduire des méthodes privées.

```
/**
 *
 */
package rational;

/**
 *
 * I implement Rational numbers.
 * Rational is defined by a couple of int
 * – numerator
 * – denominator
 *
 * @author beugnard
 *
```

```
 * @invariant non nul denominator : denominator != 0
 * @invariant positive denominator : denominator > 0
 * @invariant irreductible : gcd(|numerator|, |denominator|) = 1
 *
 * Design choice
 * – immutability for results of operations
 * – mutability to ensure invariants
 */
public class RationalFull {

  private int numerator;
  private int denominator;

  public RationalFull(int n, int d) {
    numerator = n;
    denominator = (d==0?1:d); // ensure @invariant non nul denominator
    canonical(); // ensure @invariant positive denominator and irreductible
  }
  /**
   * I return a String numerator (if denominator == 1) or numerator/denominator
   */
  @Override
  public String toString() {
    return numerator + (denominator == 1?"":"/" + denominator);
  }

  @Override
  public boolean equals(Object o) {
    if ((o == null) || !(o instanceof RationalFull)) return false;
    RationalFull r = (RationalFull) o;
    return this.numerator * r.denominator == r.numerator * this.denominator;
  }
  /**
   *
   * @param r
   * @return this + r
   */
  public RationalFull sum(RationalFull r) {
    return new RationalFull(
        this.numerator * r.denominator + r.numerator * this.denominator,
        this.denominator * r.denominator).canonical();
  }

  /**
   *
   * @param r
   * @return this – r
   */
  public RationalFull diff(RationalFull r) {
    return this.sum(r.neg());
  }
```

```java
/**
 *
 * @param r
 * @return this * r
 */
public RationalFull times(RationalFull r) {
  return new RationalFull(
      this.numerator * r.numerator,
      this.denominator * r.denominator).canonical();
}

/**
 *
 * @return -this
 */
public RationalFull neg() {
  return new RationalFull(
      - this.numerator,
      this.denominator).
      canonical(); // probably useless
}

/*
 * Function that modifies on place the form of the instance.
 * Returns a Rational to allows method calls.
 */
private RationalFull canonical() {
  if (numerator == 0) denominator = 1;
  reduce();
    canonicalSign();
    return this;
}

// internal returns void on purpose to avoid method call.
// use canonical()
private void reduce() {
  int r = gcd(Math.abs(numerator),Math.abs(denominator));
  numerator=numerator/r;
  denominator=denominator/r;
}

// internal returns void on purpose to avoid method call.
// use canonical()
private void canonicalSign() {
    if(denominator<0){
      numerator=-numerator;
        denominator=-denominator;
        }
}
```

```
  // utility
  private static int gcd(int a, int b) {
    for(int i = Math.min(a, b); i > 0; i--){
          if(a % i == 0 && b % i == 0){
              return i;
          }
      }
    return 1;// never reach but make function compilable
  }

  /**
   * @param args
   */
  public static void main(String[] args) {
    RationalFull r0, r1, r2, r3 , r4;
    r0 = new RationalFull(10,0);
    System.out.println("r0 = " + r0);
    r1 = new RationalFull(0,6);
    System.out.println("r1 = " + r1);
    r2 = new RationalFull(-2,3);
    System.out.println("r2 = " + r2);
    r3 = new RationalFull(1,2);
    System.out.println("r3 = " + r3);
    System.out.println("r1 = r2 ? " + r1.equals(r2));
    System.out.println("r1 = r3 ? " + r1.equals(r3));
    System.out.println("r1+r2 = " + (r1.sum(r2)));
    r4 = new RationalFull(-8,-4);
    System.out.println("r4 = " + r4);
    System.out.println("r1 * r2 = " + r1.times(r2));
    System.out.println("-(r3 * r4) = " + (r3.times(r4)).neg());
    System.out.println("r3 * r3 = " + (r3.times(r3)));
  }

}
```

Une variante.

```
/**
 *
 */
package rational;

/**
 * I implement Rational numbers.
 * Rational is defined by a couple of int
 * - numerator
 * - denominator
 *
 * @author beugnard
 * @author jcroyer variant
 * @invariant non nul denominator : denominator != 0
 * @invariant positive denominator : denominator > 0
```

```
 * @invariant irreductible : gcd(|numerator|, |denominator|) = 1
 */
public class RationalFull2 {

  /** The numerator. */
  private int numerator;

  /** The denominator. */
  private int denominator;

  /**
   * Constructor
   * I do most of the difficult job.
   *
   * @param n the n
   * @param d the d
   */
  public RationalFull2(int n, int d) {
    // check some annoying cases
    if (n == 0) {
      numerator = 0;
      denominator = 1;
    } else {
      if (d == 0) {
      d = 1;
      } else if (d < 0) {
        d = -d;
        n = -n;
      }
      // make it irreductible
      int r = gcd(n, d);
      n = n / r;
      d = d / r;
      // build it
      numerator = n;
      denominator = d;
    }
  }

  /**
   * I return a String numerator (if denominator == 1) or numerator/denominator.
   *
   * @return the string
   */
  @Override
  public String toString() {
    return numerator + (denominator == 1?"":"/" + denominator);
  }

  /**
   * Equals.
```

```
   *
   * @param o the o
   * @return true, if successful
   */
  @Override
  public boolean equals(Object o) {
    if ((o == null) || !(o instanceof RationalFull2)) return false;
    RationalFull2 r = (RationalFull2) o;
    //return this.numerator * r.denominator == r.numerator * this.denominator;
    // mais peut mieux faire
    return this.numerator == r.numerator && this.denominator == r.denominator;
  }

  /**
   * Sum.
   *
   * @param r the r
   * @return this + r
   */
  public RationalFull2 sum(RationalFull2 r) {
    return new RationalFull2(
        this.numerator * r.denominator + r.numerator * this.denominator,
        this.denominator * r.denominator);
  }

  /**
   * Diff.
   *
   * @param r the r
   * @return this - r
   */
  public RationalFull2 diff(RationalFull2 r) {
    return this.sum(r.neg());
  }

  /**
   * Times.
   *
   * @param r the r
   * @return this * r
   */
  public RationalFull2 times(RationalFull2 r) {
    return new RationalFull2(
        this.numerator * r.numerator,
        this.denominator * r.denominator);
  }

  /**
   * Negation
   *
   * @return -this
```

```
   */
  public RationalFull2 neg() {
    return new RationalFull2(- this.numerator, this.denominator);
  }

  /**
   * Compute the gcd of n and d.
   * @param n is > 0
   * @param d is > 0
   * @return gcd(n, d)
   */
  private static int gcd(int n, int d) {
    int a = Math.abs(n);
    int b = Math.abs(d);
    while (a != b) {
      if (a > b) {
        a = a - b;
      } else {
        b = b - a;
      }
    }
    return a;
  }

  /**
   * The main method.
   *
   * @param args the arguments
   */
  public static void main(String[] args) {
// System.out.println(" = " + gcd(2, 3));
// System.out.println(" = " + gcd(3, 2));
// System.out.println(" = " + gcd(124, 2));
// System.out.println(" = " + gcd(124, 4));
// System.out.println(" = " + gcd(31*60, 13*60));

    RationalFull2 r0, r1, r2, r3 , r4;
    r0 = new RationalFull2(10,0);
    System.out.println("r0 = " + r0);
    r1 = new RationalFull2(0,6);
    System.out.println("r1 = " + r1);
    r2 = new RationalFull2(-2,3);
    System.out.println("r2 = " + r2);
    r3 = new RationalFull2(1,2);
    System.out.println("r3 = " + r3);
    System.out.println("r1 = r2 ? " + r1.equals(r2));
    System.out.println("r1 = r3 ? " + r1.equals(r3));
    System.out.println("r1+r2 = " + (r1.sum(r2)));
    r4 = new RationalFull2(-8,-4);
    System.out.println("r4 = " + r4);
    System.out.println("r1 * r2 = " + r1.times(r2));
```

```
    System.out.println("-(r3 * r4) = " + (r3.times(r4)).neg());
    System.out.println("r3 * r3 = " + (r3.times(r3)));
  }

}
```

Exercice 2 (*Javadoc*)

Le langage Java permet d'ajouter des commentaires qui serviront à générer la documentation (`html`) de vos programmes.

▷ **Question 2.1 :**
Ajoutez des commentaires Javadoc à la classe `Rational` puis à chacune de ses méthodes et générer la documentation.

Pour aller plus loin :
- How to Write Doc Comments for the Javadoc Tool : `https://www.oracle.com/technetwork/java/javase/documentation/index-137868.html`
- Présentation de la Javadoc : `https://openclassrooms.com/fr/courses/1115306-presentation-de-la-javadoc`

Exercice 3 (*debug*)

Nous allons observer l'exécution de votre programme à l'aide du débogueur.

▷ **Question 3.1 :**
Ajoutez un point d'arrêt à l'entrée du `main`. Lancez l'exécution en mode *debug*.

▷ **Question 3.2 :**
Observez le mode de contrôle pas-à-pas du débogueur et la possibilité d'observer l'état de l'exécution. Commentez et observez les actions :
- ***Step over***
- ***Step into***
- ***Step return***
- ***Les variables locales, les arguments, this.***

Commentez et observez :
- *Step over* ; exécute l'action ; s'arrête à la suivante ; le sommet de pile change.
- *Step into* ; entre dans le corps de l'action ; la pile grandit.
- *Step return* ; exécute la méthode en cours et sort ; la pile se réduit.
- *Les variables locales, les arguments, this.* ; observe l'état des objets ; peut changer des valeurs...