Les exceptions

Exercice 1 : La classe Polynome

Dans la classe *Polynome*, il peut se produire un cas exceptionnel : le cas où le coefficient de degré 2 du polynôme est nul. Dans ce cas, le polynôme est un polynôme de degré 1 et ne permet pas de calculer ses racines car cela produit une division par zéro.

1. Sans intégrer de gestion d'exception, montrer ce qui se produit lorsque le polynôme dont on calcule les racines est de degré 1.

Le polynôme lève une exception de la classe java.lang.ArithmeticException.

2. Prendre en considération ce cas au travers d'une exception dédiée appelée *InvalidPolynomException*.

```
package fr.utt.sit.lo02.td4;

public class InvalidPolynomException extends Exception {

    private static final long serialVersionUID = 1L;
    public InvalidPolynomException(String message) {
      super(message);
    }
}
```

3. Identifier les méthodes qui lèvent des exceptions dues à des divisions par zéro. Permettre à ces méthodes de propager cette exception. Attraper cette exception dans la méthode *getRoots()* et la chainer avec une nouvelle exception du type *InvalidPolynomException*. Attraper cette nouvelle exception dans la méthode *main()* et mettre en évidence le chainage. Tester au passage les méthodes *getMessage()* et *printStackTrace()*.

```
package fr.utt.sit.lo02.td.td4;

public class PolynomeDegreDeux {

    // x2, x1, x0
    private int[] polynome;

    public PolynomeDegreDeux(int x2, int x1, int x0) {
      polynome = new int[3];
      polynome[0] = x2;
      polynome[1] = x1;
      polynome[2] = x0;
    }

    public double delta() {
      return polynome[1] * polynome[1] - 4 * polynome[0] * polynome[2];
    }

    public boolean hasComplexRoots() {
      return (this.delta() < 0);
    }

    public boolean hasDoubleRoot() {
      return (this.delta() == 0);
    }
```

```
    public boolean hasRealRoots() {
      return (this.delta() > 0);
    }

    private double moinsBSur2A () throws ArithmeticException {
      return -polynome[1] / (2 * polynome[0]);
    }

    private double racineDeltaSur2A () throws ArithmeticException {
      return Math.sqrt(this.delta()) / (2 * polynome[0]);
    }

    public int getX2() {
      return polynome[0];
    }

    public int getX1() {
      return polynome[1];
    }

    public int getX0() {
      return polynome[2];
    }

    public boolean equals(Object o) {
      if (o instanceof PolynomeDegreDeux) {
          PolynomeDegreDeux p = (PolynomeDegreDeux)o;
          return ((p.getX2() == this.getX2()) & (p.getX1() == this.getX1())
& (p.getX0() == this.getX0()));
      } else {
          return false;
      }

    }

    public Complexe[] getRoots() throws InvalidPolynomException {

      Complexe[] racines;

      try {
          if (this.hasDoubleRoot()) {
            racines = new Complexe[1];
            racines[0] = new Complexe (this.moinsBSur2A(), 0);
          } else {
            racines = new Complexe[2];
            if (this.hasRealRoots()) {
                racines[0] = new Complexe(this.moinsBSur2A() +
this.racineDeltaSur2A(), 0);
                racines[1] = new Complexe(this.moinsBSur2A() -
this.racineDeltaSur2A(), 0);
            } else {
                racines[0] = new Complexe(this.moinsBSur2A(),
this.racineDeltaSur2A());
                racines[1] = racines[0].conjugue();
            }
          }
      } catch (ArithmeticException e) {
          InvalidPolynomException ipe = new InvalidPolynomException("Not a
second order polynom: " + this);
          ipe.initCause(e);
          throw ipe;
      }
```

```
      return racines;
    }

    public String toString() {
      return new String(polynome[0] + "x2 + " + polynome[1] + "x + " +
polynome[2]);
    }

    public static void main(String[] args) {

      PolynomeDegreDeux p1 = new PolynomeDegreDeux (2, -1, 1);

      try {
          System.out.println("Racines : " + p1.getRoots());
      } catch (InvalidPolynomException e) {
          e.printStackTrace();
      }

    }
}
```

Exercice 2 : La classe Lampe

On désire faire évoluer la classe lampe pour prendre en compte deux nouveaux aspects :

- la durée de vie de la lampe. Une lampe ne peut être allumée qu'un certain nombre de fois. Au-delà, le filament se rompt et la lampe ne peut plus être allumée.
- Le fait qu'une lampe dans un état ne puisse être à nouveau placée dans le même état. Par exemple, une lampe allumée ne doit pas pouvoir être allumée à nouveau. Il est de même pour l'extinction.

1. Intégrer ces fonctionnalités dans la classe Lampe à l'aide d'exceptions dédiées.

```
package fr.utt.sit.lo02.td4.lampe;

public class EtatInvalideException extends Exception {

    private static final long serialVersionUID = 1L;

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

```
package fr.utt.sit.lo02.td4.lampe;

public class LampeGrilleeException extends Exception {

    private static final long serialVersionUID = 1L;

}
```

```
package fr.utt.sit.lo02.td4.lampe;

public class Lampe {

    public final static int PUISSANCE_STANDARD = 100;
    public final static int NOMBRE_UTILISATION_MAXIMUM = 5;

    private int puissance;
```

```
    private boolean allumee;
    private int nombreUtilisations;

    public Lampe(int puissance) {
      this.puissance = puissance;
      this.allumee = false;
      this.nombreUtilisations = 0;
    }

    public int getPuissance() {
      return puissance;
    }

    public boolean isAllumee() {
      return allumee;
    }

    public void allumer() throws EtatInvalideException,
LampeGrilleeException{
      if (this.allumee == true) {
          throw new EtatInvalideException("La lampe est déjà allumée");
      } else {
          if (nombreUtilisations < Lampe.NOMBRE_UTILISATION_MAXIMUM) {
          this.allumee = true;
          this.nombreUtilisations++;
          } else {
            throw new LampeGrilleeException();
          }
      }
    }

    public void eteindre() throws EtatInvalideException {
      if (this.allumee == false) {
          throw new EtatInvalideException("La lampe est déjà éteinte");
      } else {
          this.allumee = false;
      }
    }

    public String toString() {
      StringBuffer sb = new StringBuffer();
      sb.append("Lampe (");
      sb.append("Puissance : " + puissance);
      sb.append("\t");
      sb.append("Allumée : " + allumee);
      sb.append(")\n");
      return sb.toString();
    }
}
```

2. Le commutateur est l'élément qui effectue les appels aux méthodes de la classe Lampe, et à ce titre récupère les exceptions levées par cette dernière. Montrer comment il peut les attraper, les propager, ou en propager de nouvelles par le biais du chainage des exceptions.

```
package fr.utt.sit.lo02.td4.lampe;

public class CommutationImpossibleException extends Exception {

    private static final long serialVersionUID = 1L;
```

```
}
```

```
package fr.utt.sit.lo02.td4.lampe;

public class Commutateur {

    private Lampe[] lampes;
    private int etat;
    private int nombreEtats;

    public Commutateur (int nombreLampes) {

      etat = 0;
      nombreEtats = nombreLampes * 2;
      lampes = new Lampe[nombreLampes];

      Lampe lampeFaible = new Lampe (Lampe.PUISSANCE_STANDARD / 2);
      lampes[0] = lampeFaible;

      for (int i = 1; i < nombreLampes; i++) {
          Lampe lampeStandard = new Lampe (Lampe.PUISSANCE_STANDARD);
          lampes[i] = lampeStandard;
      }
    }

    public void commuter() throws CommutationImpossibleException {
      etat = (etat + 1) % nombreEtats;

      int puissanceRequise = etat * Lampe.PUISSANCE_STANDARD / 2;

      try {
      // Allumage/eteignage de la lampe faible
      Lampe lampeFaible = lampes[0];

      if ((puissanceRequise % Lampe.PUISSANCE_STANDARD) != 0) {
          lampeFaible.allumer();
      } else {
          lampeFaible.eteindre();
      }

      // Allumage/Eteignage des autres lampes
      for (int i = 1; i <= (puissanceRequise / Lampe.PUISSANCE_STANDARD);
i++) {
          Lampe lampe = lampes[i];
          lampe.allumer();
      }

      for (int i = Math.max((puissanceRequise / Lampe.PUISSANCE_STANDARD) +
1, 1); i < lampes.length; i++) {
          Lampe lampe = lampes[i];
          lampe.eteindre();
      }
      } catch (EtatInvalideException e) {
          System.out.println("Attention : " + e.getMessage());
      } catch (LampeGrilleeException e) {
          CommutationImpossibleException cie = new
CommutationImpossibleException();
          cie.initCause(e);
          throw cie;
      }
    }
```

```
    public String toString() {
      StringBuffer sb = new StringBuffer();
      sb.append("Commutateur (");
      sb.append("Etat : " + etat);
      sb.append(")\n");
      for (int i = 0; i < lampes.length; i++) {
          sb.append(lampes[i]);
      }
      return sb.toString();
    }
}
```

```
package fr.utt.sit.lo02.td4.lampe;

public class Interrupteur {
    private Commutateur commutateur;

    public Interrupteur () {
      commutateur = new Commutateur(5);
    }

    public void appuyer() {
      try {
          commutateur.commuter();
      } catch (Exception e) {
          e.printStackTrace();
      }
    }

    public String toString() {
      StringBuffer sb = new StringBuffer();
      sb.append("Interrupteur\n");
      sb.append(commutateur);
      return sb.toString();
    }

    public static void main (String[] args) {

      Interrupteur inter = new Interrupteur();

      for (int i = 0; i < 20; i++) {
          inter.appuyer();
          System.out.println(inter);
      }
    }
}
```