UV FIL1 "Langage de programmation"
Modularité et typage
Travaux pratiques
Architectures en couches pour les complexes

Hervé Grall
Département d'informatique
École des mines de Nantes

2013/2014

Ce TP a pour but de créer plusieurs constructions alternatives d'un type de données, les complexes, en utilisant les méthodes et techniques vues en cours, lors de la séance dédiée au polymorphisme et la réutilisation modulaire. Partant de l'interface `Complexe`, on commence par élaborer une hiérarchie d'interfaces suivant les recommandations vues en cours.

- Exprimer par la hiérarchie la relation « est un ».
- Généraliser la relation « est un » pour intégrer à la hiérarchie les concepts pertinents.
- Veiller à la complétude et à la minimalité des interfaces (toutes les opérations utiles, rien que les opérations utiles).
- Rendre le code polymorphe en utilisant le plus possible les interfaces les plus générales.

Dans un second temps, on construit les classes d'implémentation à partir d'une architecture à deux niveaux et suivant les quatre méthodes vues en cours.

- Agrégation avec une seule classe
- Agrégation avec deux classes
- Héritage avec une construction *bottom up*
- Héritage avec une construction *top down*

Ces quatre méthodes constituent des alternatives. Pour les trois dernières, elles permettent de construire une classe d'implémentation des services d'un des niveaux à partir d'une classe d'implémentation des services de l'autre niveau. Pratiquement, on choisit une de ces méthodes, en fonction des classes d'implémentation déjà existantes, ou des classes qu'on souhaite réutiliser.

1 La hiérarchie d'interfaces

Nous partons de l'interface `Complexe` suivante.

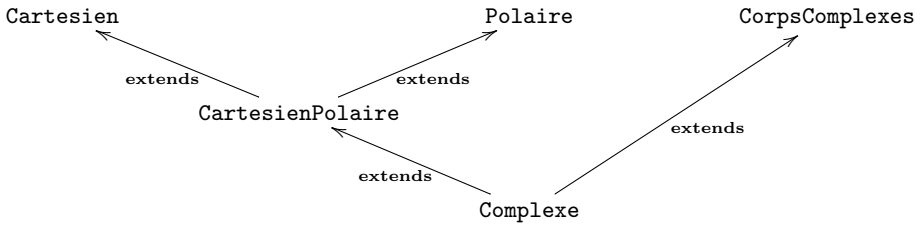
```
public interface Complexe {
    /* Accesseurs */
    public double getX();
```

```
    public double getY();
    public double getRho();
    public Angle getTheta();

    /* Fabriques */
    public Complexe creer(double x, double y);
    public Complexe creer(double rho, Angle theta);

    /* Services */
    public Complexe somme(Complexe c);
    public Complexe produit(Complexe c);
}
```

Il est alors possible de la décomposer suivant la nature des méthodes et suivant le point de vue adopté pour accéder au complexe.

- Accesseurs – Point de vue cartésien
  Interface `Cartesien` contenant les méthodes `getX` et `getY`
- Accesseurs – Point de vue polaire
  Interface `Polaire` contenant les méthodes `getRho` et `getTheta`
- Opérations algébriques
  Interface `CorpsComplexes` contenant les méthodes opératoires `somme` et `produit`

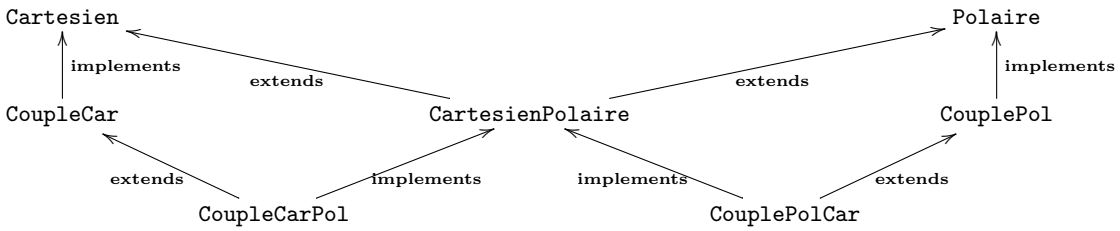
Finalement, on peut obtenir la hiérarchie suivante.



La classe `CartesienPolaire` hérite des classes `Cartesien` et `Polaire` sans les étendre. Quant à la classe `Complexe`, non seulement elle hérite des classes `CartesienPolaire` et `CorpsComplexes`, mais elle les étend aussi en déclarant les fabriques `creer`.

2 Coordonnées cartésiennes et polaires

On implémente les interfaces `Cartesien` et `Polaire` dans deux classes `CoupleCar` et `CouplePol`, utilisant pour attributs un couple de coordonnées cartésiennes et polaires respectivement. Puis on en déduit par héritage deux implémentations, `CoupleCarPol` et `CouplePolCar`, de l'interface `CartesienPolaire`.

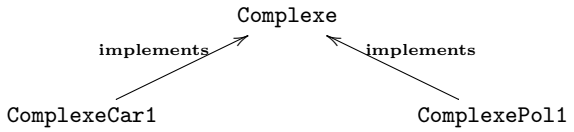


3 Implémentation des complexes

Dans la hiérarchie des interfaces introduite initialement, il est possible de distinguer deux niveaux d'abstraction : en effet, les services de l'interface `CorpsComplexes`, plus abstraits, peuvent être implémentés en utilisant ceux de l'interface `CartesienPolaire`, plus concrets. Pour implémenter l'interface `Complexe`, nous allons donc construire une architecture à deux niveaux. Plusieurs solutions sont étudiées. Précisément, dans les solutions qui suivent, certaines utilisent les classes d'implémentation `CoupleCarPol` et `CouplePolCar`. Elles démontrent qu'il est possible de réutiliser des implémentations existantes. Les autres solutions qui n'utilisent pas ces classes d'implémentation, ou bien ne cherchent pas à favoriser la réutilisation, ou bien permettent de réutiliser une autre classe d'implémentation, autrement dit celle implémentant les services déclarés dans l'interface `CorpsComplexes`.

3.1 Agrégation avec une seule classe d'implémentation

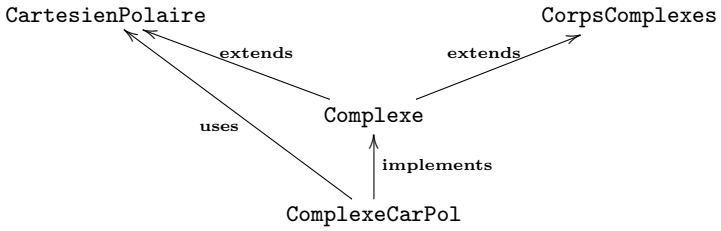
C'est la solution que nous avons vue pendant le premier TP. Les classes `ComplexeCar1` et `ComplaxePol1` implémentent directement l'interface `Complexe` sans recourir à l'héritage.



Cette première solution n'utilise pas de classe existante. Elle ne permet pas non plus de réutiliser une partie de l'implémentation, comme celle implémentant l'interface `CartesienPolaire` ou celle implémentant l'interface `CorpsComplexes`, puisqu'elles ne sont pas séparables.

3.2 Agrégation avec deux classes d'implémentation

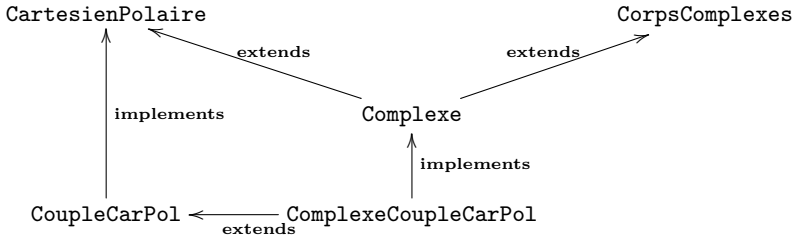
On définit une classe `ComplexeCarPol` qui implémente l'interface `Complexe` en utilisant un objet du type `CartesienPolaire`. Un complexe de cette classe contient donc un couple de coordonnées, qui peuvent être cartésiennes ou polaires.



Cette seconde solution utilise les classes implémentant `CartesienPolaire`, soit `CoupleCarPol` et `CouplePolCar`. Elle permet cependant de définir une classe implémentant l'interface `Complexe` avec une dépendance faible relativement aux classes d'implémentation `CoupleCarPol` et `CouplePolCar`, puisque celle-ci ne survient qu'à la construction des objets de la classe.

3.3 Héritage avec une construction bottom-up

Pour implémenter l'interface `Complexe`, on définit deux classes `ComplexeCoupleCarPol` et `ComplexeCouplePolCar`, héritant des classes `CoupleCarPol` et `CouplePolCar` respectivement, et correspondant au niveau concret.

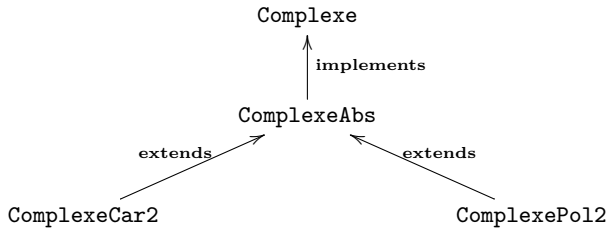


Le schéma est analogue pour la classe `ComplexeCouplePolCar`.

Cette troisième solution utilise les classes implémentant `CartesienPolaire`, soit `CoupleCarPol` et `CouplePolCar`. La dépendance est forte puisque les classes implémentant l'interface `Complexe` en héritent.

3.4 Héritage avec une construction top-down

Pour implémenter l'interface `Complexe`, on introduit une classe abstraite `ComplexeAbs`. Celle-ci n'implémente que les méthodes du niveau abstrait, autrement dit les méthodes de l'interface `CorpsComplexes` ; elle ne possède pas d'attributs ni de constructeurs. On complète ensuite cette classe abstraite dans deux classes d'implémentation `ComplexeCar2` et `ComplexePol2`, pour finalement implémenter complètement l'interface `Complexe`.



Cette quatrième et dernière solution n'utilise pas les classes implémentant `CartesienPolaire`, soit `CoupleCarPol` et `CouplePolCar`. En revanche, elle permet de réutiliser une implémentation de l'interface `CorpsComplexes`.

4 Travaux pratiques

On placera chaque classe ou interface `X` dans un fichier `X.java`. Dans chaque classe d'implémentation, spécialiser la méthode `toString` (de la classe racine `Object`) de manière à produire une chaîne significative de caractères. Par exemple, on représentera un couple de coordonnées cartésiennes x et y par la chaîne (x, y), et un couple de coordonnées polaires ρ et θ par la chaîne $[\rho, \theta]$. De même, spécialiser la méthode **boolean** `equals(Object o)`, de manière à tester l'égalité. Enfin, les données implémentées seront immutables, autrement dit en lecture seulement.

1. Récupérer l'archive sur campus. Elle contient :
   - la hiérarchie des interfaces décrite dans la section 1,
   - la classe `Angle`,
   - la classe module `Conversion`, permettant de réaliser les conversions entre les coordonnées polaires et celles cartésiennes,
   - les classes `ComplexeCar1` et `ComplexePol1` implémentant l'interface `Complexe`, et décrites dans la section 3.1,
   - les classes d'implémentation `CoupleCar`, `CouplePol`, `CoupleCarPol` et `CouplePolCar`, implémentant les interfaces représentant les coordonnées cartésiennes et polaires, et décrites dans la section 2,
   - la classe module `Comparaison`, qui permet de tester l'égalité de deux réels de type **double**, suivant une certaine précision.

   Étudier le code fourni. Pour implémenter les méthodes `equals` et `toString`, on pourra s'inspirer des exemples donnés dans les classes `CoupleCar`, `CouplePol`, `CoupleCarPol` et `CouplePolCar`.
2. Définir la classe `ComplexeCarPol` implémentant l'interface `Complexe` (cf. sect. 3.2). On utilisera les constructeurs suivants.
   - `ComplexeCarPol`(**double** `x`, **double** `y`)
   - `ComplexeCarPol`(**double** `rho`, `Angle theta`)
3. Définir les classes `ComplexeCoupleCarPol` et `ComplexeCouplePolCar`, héritant des classes `CoupleCarPol` et `CouplePolCar` respectivement, et implémentant l'interface `Complexe` (cf. sect. 3.3). On utilisera les constructeurs suivants.
   - `ComplexeCoupleCarPol`(**double** `x`, **double** `y`)
   - `ComplexeCoupleCarPol`(**double** `rho`, `Angle theta`)
   - `ComplexeCouplePolCar`(**double** `rho`, `Angle theta`)
   - `ComplexeCouplePolCar`(**double** `x`, **double** `y`)
4. Définir la classe abstraite `ComplexeAbs` implémentant partiellement l'interface `Complexe` (cf. sect. 3.4). En dériver par héritage les deux classes d'implémentation `ComplexeCar2` et `ComplexePol2`. On utilisera les constructeurs suivants.
   - `ComplexeCar2`(**double** `x`, **double** `y`)
   - `ComplexeCar2`(**double** `rho`, `Angle theta`)
   - `ComplexePol2`(**double** `rho`, `Angle theta`)
   - `ComplexePol2`(**double** `x`, **double** `y`)
5. Tester votre classe dans la fonction `main` d'une classe `Test`. Pour chaque classe d'implémentation, réaliser le scénario de test suivant.
   - Créer un complexe `a` de valeur $1 - 1\,\mathrm{i}$.
   - Afficher ses coordonnées cartésiennes.
   - Afficher ses coordonnées polaires.
   - Créer un complexe `b` de valeur $2\,\mathrm{e}^{\mathrm{i}\,\pi/2}$.
   - Afficher ses coordonnées cartésiennes.
   - Afficher ses coordonnées polaires.
   - Créer un complexe `c` égal à la somme de `a` et `b`.
   - Comparer `c` avec le complexe $1 + 1\,\mathrm{i}$.
   - Créer un complexe `d` égal au produit de `a` et `c`.
   - Comparer `d` avec le complexe 2.
6. Pour conclure, comparer les différentes solutions pour l'implémentation des complexes, en reprenant les quatre schémas et les brefs commentaires les suivant.

5 Une correction

Les deux classes modules

```
public class Conversion {
  public static double x(double rho, Angle theta){
    return rho * Math.cos(theta.getAngle());
  }
  public static double y(double rho, Angle theta){
    return rho * Math.sin(theta.getAngle());
  }
  public static double rho(double x, double y){
    return Math.sqrt(x*x + y*y);
  }
  public static Angle theta(double x, double y){
    return new Angle(Math.atan2(y, x));
  }
}

public class Comparaison {
  public static double precision = 1E-15;
  public static boolean estEgal(double x, double y){
    return (x - y) < precision;
  }
}
```

Classe `Angle`

```
public class Angle {
  private double theta; // angle en radians (dans [0, 2.PI[)
  public Angle(double angleEnRadians) {
    this.theta = angleEnRadians - 2 * Math.PI
    * Math.floor(angleEnRadians / (2 * Math.PI));
  }
  public double getAngle() {
    return this.theta;
  }
  public Angle somme(Angle a) {
    return new Angle(a.getAngle() + this.getAngle());
  }
  public Angle facteur(double k) {
    return new Angle(k * this.getAngle());
  }
  public String toString() {
    return "" + theta;
  }
  public boolean equals(Object o){
    if(!(o instanceof Angle)) return false;
    Angle a = (Angle)o;
    return Comparaison.estEgal(this.getAngle(), a.getAngle());
  }
}
```

Hiérarchie

```
public interface Cartesien {
  double getX();
  double getY();
}
public interface Polaire {
  double getRho();
  Angle getTheta();
}
public interface CartesienPolaire extends Cartesien, Polaire {}

public interface CorpsComplexes {
    public Complexe somme(Complexe c);
    public Complexe produit(Complexe c);
}

public interface Complexe extends CartesienPolaire, CorpsComplexes {
  public Complexe creer(double x, double y);
  public Complexe creer(double rho, Angle theta);
}
```

Implémentation de l'interface `CartesienPolaire`

```
public class CoupleCar implements Cartesien {
  private double x;
  private double y;
  public CoupleCar(double x, double y){
    this.x = x;
    this.y = y;
  }
  public double getX(){
    return x;
  }
  public double getY(){
    return y;
  }
  public boolean equals(Object o){
    if(!(o instanceof Cartesien)) return false;
    Cartesien c = (Cartesien)o;
    return Comparaison.estEgal(this.getX(), c.getX())
    && Comparaison.estEgal(this.getY(), c.getY());
  }
  public String toString(){
    return "(" + this.getX() + ", " + this.getY() + ")";
  }
}
public class CoupleCarPol extends CoupleCar implements CartesienPolaire {
  public CoupleCarPol(double x, double y){
    super(x, y);
  }
  public CoupleCarPol(double rho, Angle theta){
    super(Conversion.x(rho, theta), Conversion.y(rho, theta));
  }
  public double getRho(){
    return Conversion.rho(this.getX(), this.getY());
  }
```

```
  public Angle getTheta(){
    return Conversion.theta(this.getX(), this.getY());
  }
  public boolean equals(Object o){
    if(!(o instanceof CartesienPolaire)) return false;
    CartesienPolaire c = (CartesienPolaire)o;
    return Comparaison.estEgal(this.getX(), c.getX())
    && Comparaison.estEgal(this.getY(), c.getY());
  }
}
public class CouplePol implements Polaire {
  private double rho;
  private Angle theta;
  public CouplePol(double rho, Angle theta){
    this.rho = rho;
    this.theta = theta;
  }
  public double getRho(){
    return rho;
  }
  public Angle getTheta(){
    return theta;
  }
  public boolean equals(Object o){
    if(!(o instanceof Polaire)) return false;
    Polaire c = (Polaire)o;
    return Comparaison.estEgal(this.getRho(), c.getRho())
    && (this.getTheta().equals(c.getTheta()));
  }
  public String toString(){
    return "[" + this.getRho()
    + ", " + this.getTheta() + "]";
  }
}
public class CouplePolCar extends CouplePol implements CartesienPolaire {
  public CouplePolCar(double x, double y){
    super(Conversion.rho(x, y), Conversion.theta(x, y));
  }
  public CouplePolCar(double rho, Angle theta){
    super(rho, theta);
  }
  public double getX(){
    return Conversion.x(this.getRho(), this.getTheta());
  }
  public double getY(){
    return Conversion.y(this.getRho(), this.getTheta());
  }
  public boolean equals(Object o){
    if(!(o instanceof CartesienPolaire)) return false;
    CartesienPolaire c = (CartesienPolaire)o;
    return Comparaison.estEgal(this.getRho(), c.getRho())
    && (this.getTheta().equals(c.getTheta()));
  }
}
```

Implémentation de l'interface `Complexe`

```
public class ComplexeCar1 implements Complexe {
  private double x;
  private double y;

  public ComplexeCar1(double x, double y){
    this.x = x;
    this.y = y;
  }
  public ComplexeCar1(double rho, Angle theta){
    this.x = Conversion.x(rho, theta);
    this.y = Conversion.y(rho, theta);
  }

  public double getX(){
    return x;
  }
  public double getY(){
    return y;
  }
  public double getRho(){
    return Conversion.rho(this.getX(), this.getY());
  }
  public Angle getTheta(){
    return Conversion.theta(this.getX(), this.getY());
  }

  public Complexe creer(double x, double y){
    return new ComplexeCar1(x, y);
  }
  public Complexe creer(double rho, Angle theta){
    return new ComplexeCar1(rho, theta);
  }

  public Complexe somme(Complexe c){
    return this.creer(this.getX() + c.getX(),
    this.getY() + c.getY());
  }
  public Complexe produit(Complexe c){
    return this.creer(this.getRho() * c.getRho(),
    this.getTheta().somme(c.getTheta()));
  }

  public boolean equals(Object o){
    if(!(o instanceof Complexe)) return false;
    Complexe c = (Complexe)o;
    return Comparaison.estEgal(this.getX(), c.getX())
   && Comparaison.estEgal(this.getY(), c.getY());
  }
  public String toString(){
    return "(" + this.getX() + ", " + this.getY() + ")";
  }
}
```

```
public class ComplexePol1 implements Complexe {
  private double rho;
  private Angle theta;

  public ComplexePol1(double x, double y){
    this.rho = Conversion.rho(x, y);
    this.theta = Conversion.theta(x, y);
  }
  public ComplexePol1(double rho, Angle theta){
    this.rho = rho;
    this.theta = theta;
  }

  public double getX(){
    return Conversion.x(this.getRho(), this.getTheta());
  }
  public double getY(){
    return Conversion.y(this.getRho(), this.getTheta());
  }
  public double getRho(){
    return rho;
  }
  public Angle getTheta(){
    return theta;
  }

  public Complexe creer(double x, double y){
    return new ComplexePol1(x, y);
  }
  public Complexe creer(double rho, Angle theta){
    return new ComplexePol1(rho, theta);
  }

  public Complexe somme(Complexe c){
    return this.creer(this.getX() + c.getX(),
    this.getY() + c.getY());
  }
  public Complexe produit(Complexe c){
    return this.creer(this.getRho() * c.getRho(),
    this.getTheta().somme(c.getTheta()));
  }

  public boolean equals(Object o){
    if(!(o instanceof Complexe)) return false;
    Complexe c = (Complexe)o;
    return Comparaison.estEgal(this.getRho(), c.getRho())
    && (this.getTheta().equals(c.getTheta()));
  }
  public String toString(){
    return "[" + this.getRho()
    + ", " + this.getTheta() + "]";
  }
}
```

```
public class ComplexeCarPol implements Complexe {
  private CartesienPolaire c;
  public ComplexeCarPol(double x, double y){
    c = new CoupleCarPol(x, y);
  }
  public ComplexeCarPol(double rho, Angle theta){
    c = new CouplePolCar(rho, theta);
  }
  public double getX(){
    return c.getX();
  }
  public double getY(){
    return c.getY();
  }
  public double getRho(){
    return c.getRho();
  }
  public Angle getTheta(){
    return c.getTheta();
  }

  public Complexe creer(double x, double y){
    return new ComplexeCarPol(x, y);
  }
  public Complexe creer(double rho, Angle theta){
    return new ComplexeCarPol(rho, theta);
  }

  public Complexe somme(Complexe c){
    return this.creer(this.getX() + c.getX(),
    this.getY() + c.getY());
  }
  public Complexe produit(Complexe c){
    return this.creer(this.getRho() * c.getRho(),
    this.getTheta().somme(c.getTheta()));
  }

  public boolean equals(Object o){
    if(!(o instanceof Complexe)) return false;
    Complexe c = (Complexe)o;
    return Comparaison.estEgal(this.getX(), c.getX())
    && Comparaison.estEgal(this.getY(), c.getY());
  }
  public String toString(){
    return "(" + this.getX() + ", " + this.getY() + ")";
  }
}

public class ComplexeCoupleCarPol extends CoupleCarPol implements Complexe {
  public ComplexeCoupleCarPol(double x, double y){
    super(x, y);
  }
  public ComplexeCoupleCarPol(double rho, Angle theta){
    super(rho, theta);
```

```
  }

  public Complexe creer(double x, double y){
    return new ComplexeCoupleCarPol(x, y);
  }
  public Complexe creer(double rho, Angle theta){
    return new ComplexeCoupleCarPol(rho, theta);
  }

  public Complexe somme(Complexe c){
    return this.creer(this.getX() + c.getX(),
    this.getY() + c.getY());
  }
  public Complexe produit(Complexe c){
    return this.creer(this.getRho() * c.getRho(),
    this.getTheta().somme(c.getTheta()));
  }

  public boolean equals(Object o){
    if(!(o instanceof Complexe)) return false;
    Complexe c = (Complexe)o;
    return Comparaison.estEgal(this.getX(), c.getX())
    && Comparaison.estEgal(this.getY(), c.getY());
  }
}

public class ComplexeCouplePolCar extends CouplePolCar implements Complexe {
  public ComplexeCouplePolCar(double x, double y){
    super(x, y);
  }
  public ComplexeCouplePolCar(double rho, Angle theta){
    super(rho, theta);
  }

  public Complexe creer(double x, double y){
    return new ComplexeCouplePolCar(x, y);
  }
  public Complexe creer(double rho, Angle theta){
    return new ComplexeCouplePolCar(rho, theta);
  }

  public Complexe somme(Complexe c){
    return this.creer(this.getX() + c.getX(),
    this.getY() + c.getY());
  }
  public Complexe produit(Complexe c){
    return this.creer(this.getRho() * c.getRho(),
    this.getTheta().somme(c.getTheta()));
  }

  public boolean equals(Object o){
    if(!(o instanceof Complexe)) return false;
    Complexe c = (Complexe)o;
    return Comparaison.estEgal(this.getRho(), c.getRho())
```

```
    && (this.getTheta().equals(c.getTheta()));
  }
}

public abstract class ComplexeAbs implements Complexe {
  public Complexe somme(Complexe c){
    return this.creer(this.getX() + c.getX(),
    this.getY() + c.getY());
  }
  public Complexe produit(Complexe c){
    return this.creer(this.getRho() * c.getRho(),
    this.getTheta().somme(c.getTheta()));
  }

  public boolean equals(Object o){
    if(!(o instanceof Complexe)) return false;
    Complexe c = (Complexe)o;
    return Comparaison.estEgal(this.getX(), c.getX())
    && Comparaison.estEgal(this.getY(), c.getY());
  }
  public String toString(){
    return "(" + this.getX() + ", " + this.getY() + ")";
  }
}

public class ComplexeCar2 extends ComplexeAbs implements Complexe {
  private double x;
  private double y;
  public ComplexeCar2(double x, double y){
    this.x = x;
    this.y = y;
  }
  public ComplexeCar2(double rho, Angle theta){
    this.x = Conversion.x(rho, theta);
    this.y = Conversion.y(rho, theta);
  }
  public double getX(){
    return x;
  }
  public double getY(){
    return y;
  }
  public double getRho(){
    return Conversion.rho(this.getX(), this.getY());
  }
  public Angle getTheta(){
    return Conversion.theta(this.getX(), this.getY());
  }
  public Complexe creer(double x, double y){
    return new ComplexeCar2(x, y);
  }
  public Complexe creer(double rho, Angle theta){
    return new ComplexeCar2(rho, theta);
  }
```

```
}
public class ComplexePol2 extends ComplexeAbs implements Complexe {
  private double rho;
  private Angle theta;
  public ComplexePol2(double x, double y){
    this.rho = Conversion.rho(x, y);
    this.theta = Conversion.theta(x, y);
  }
  public ComplexePol2(double rho, Angle theta){
    this.rho = rho;
    this.theta = theta;
  }
  public double getX(){
    return Conversion.x(this.getRho(), this.getTheta());
  }
  public double getY(){
    return Conversion.y(this.getRho(), this.getTheta());
  }
  public double getRho(){
    return rho;
  }
  public Angle getTheta(){
    return theta;
  }
  public Complexe creer(double x, double y){
    return new ComplexePol2(x, y);
  }
  public Complexe creer(double rho, Angle theta){
    return new ComplexePol2(rho, theta);
  }
}
```

La classe de test

```
public class Test {
  private static int score = 0;
  private static int total = 0;
  public static void main(String[] args){
    Complexe fab = new ComplexeCar1(0, 0);
    tester(fab);
    fab = new ComplexePol1(0, 0);
    tester(fab);
    fab = new ComplexeCarPol(0, 0);
    tester(fab);
    fab = new ComplexeCoupleCarPol(0, 0);
    tester(fab);
    fab = new ComplexeCouplePolCar(0, 0);
    tester(fab);
    fab = new ComplexeCar2(0, 0);
    tester(fab);
    fab = new ComplexePol2(0, 0);
    tester(fab);
    System.out.println("-----------------------------------------");
    System.out.println("Score final : " + score + " / " + total);
  }
```

```
  private static void tester(Complexe fab){
    Complexe a = fab.creer(1,-1);
    testerCoordonnees(a, 1.0, -1.0, Math.sqrt(2.0), (7.0 * Math.PI / 4.0));

    Complexe b = fab.creer(2, new Angle(Math.PI/2.0));
    testerCoordonnees(b, 0.0, 2.0, 2.0, (Math.PI / 2.0));

    Complexe c = a.somme(b);
    testerEgalite(c, fab.creer(1.0, 1.0));
    testerCoordonnees(c, 1.0, 1.0, Math.sqrt(2.0), Math.PI/ 4.0);

    Complexe d = a.produit(c);
    testerEgalite(d, fab.creer(2.0, 0.0));
    testerCoordonnees(d, 2.0, 0.0, 2.0, 0.0);
  }

  private static void testerCoordonnees(Complexe c,
    double x, double y,
    double rho, double theta){
    total++;
    if(Comparaison.estEgal(c.getX(), x)){
      score++;
    }else{
      System.out.println("-----------------------------------------------");
      System.out.println("Erreur pour le complexe " + c + " : "
        + c.getClass());
      System.out.println("x = " + c.getX()
        + " au lieu de " + x);
    }
    total++;
    if(Comparaison.estEgal(c.getY(), y)){
      score++;
    }else{
      System.out.println("-----------------------------------------------");
      System.out.println("Erreur pour le complexe " + c + " : "
        + c.getClass());
      System.out.println("y = " + c.getY()
        + " au lieu de " + y);
    }
    total++;
    if(Comparaison.estEgal(c.getRho(), rho)){
      score++;
    }else{
      System.out.println("-----------------------------------------------");
      System.out.println("Erreur pour le complexe " + c + " : "
        + c.getClass());
      System.out.println("rho = " + c.getRho()
        + " au lieu de " + rho);
    }
    total++;
    if(Comparaison.estEgal(c.getTheta().getAngle(), theta)){
      score++;
    }else{
      System.out.println("-----------------------------------------------");
```

```
      System.out.println("Erreur pour le complexe " + c + " : "
        + c.getClass());
      System.out.println("theta = " + c.getTheta().getAngle()
        + " au lieu de " + theta);
    }
  }
  private static void testerEgalite(Complexe c1, Complexe c2){
    total++;
    if(c1.equals(c2)){
      score++;
    }else{
      System.out.println("-----------------------------------------------");
      System.out.println("Erreur pour le test d'egalite : "
      + c1 + " : " + c1.getClass()
      + " egal a " + c2 + " : " + c2.getClass() );
    }
  }
}
```