

Application à la POO – ARIVE – 1

Christophe BLANC

IUT MONTLUÇON
UNIVERSITÉ Blaise Pascal

Janvier 2016

- ① n.f. – 1922 ; arg. *faire la java* (1901), « danser en remuant les épaules » ; o.i. ◇ Danse de bal musette à trois temps, assez rapide – Air, musique qui l'accompagne. ◇ Loc. Fam. *Faire la java, faire la foire*. V. noce, nouba.
- ② Ile volcanique, Indonésie, longue de 1000 km et large de 190 km au max., s'allongeant d'O. en E. entre les îles de Sumatra et de Bali.
- ③ Mot sujet à nombre jeux de mots. *Java Swinger, Java jaser* ou *Java bien* en sont des exemples.
- ④ Langage de programmation orientée objet créé par Sun en 1994.

- 1 n.f. – 1922 ; arg. *faire la java* (1901), « danser en remuant les épaules » ; o.i. ◇ Danse de bal musette à trois temps, assez rapide – Air, musique qui l'accompagne. ◇ Loc. Fam. *Faire la java, faire la foire*. V. noce, nouba.
- 2 Ile volcanique, Indonésie, longue de 1000 km et large de 190 km au max., s'allongeant d'O. en E. entre les îles de Sumatra et de Bali.
- 3 Mot sujet à nombre jeux de mots. *Java Swinger, Java jaser* ou *Java bien* en sont des exemples.
- 4 Langage de programmation orientée objet créé par Sun en 1994.

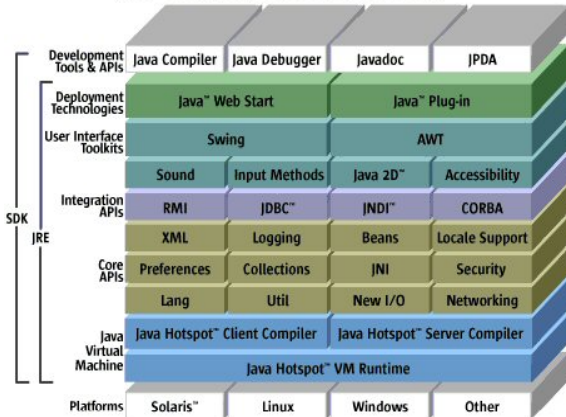
- ① n.f. – 1922 ; arg. *faire la java* (1901), « danser en remuant les épaules » ; o.i. ◇ Danse de bal musette à trois temps, assez rapide – Air, musique qui l'accompagne. ◇ Loc. Fam. *Faire la java, faire la foire*. V. noce, nouba.
- ② Ile volcanique, Indonésie, longue de 1000 km et large de 190 km au max., s'allongeant d'O. en E. entre les îles de Sumatra et de Bali.
- ③ Mot sujet à nombre jeux de mots. *Java Swinger, Java jaser* ou *Java bien* en sont des exemples.
- ④ Langage de programmation orientée objet créé par Sun en 1994.

- ① n.f. – 1922 ; arg. *faire la java* (1901), « danser en remuant les épaules » ; o.i. ◇ Danse de bal musette à trois temps, assez rapide – Air, musique qui l'accompagne. ◇ Loc. Fam. *Faire la java, faire la foire*. V. noce, nouba.
- ② Ile volcanique, Indonésie, longue de 1000 km et large de 190 km au max., s'allongeant d'O. en E. entre les îles de Sumatra et de Bali.
- ③ Mot sujet à nombre jeux de mots. *Java Swinger, Java jaser* ou *Java bien* en sont des exemples.
- ④ Langage de programmation orientée objet créé par Sun en 1994.

Rappels généraux

Le JDK 1.4

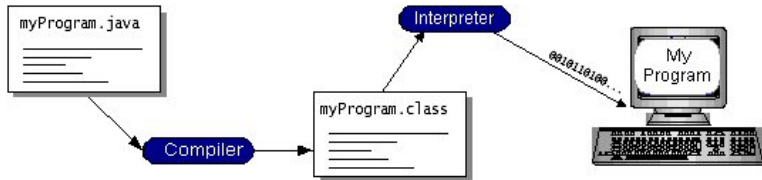
Java™ 2 Platform, Standard Edition v 1.4



👉 Compilation d'un programme Java : génération de byte-code

👉 Le byte code est :

- proche d'un langage machine
- indépendant de l'environnement d'exécution (matériel + OS)



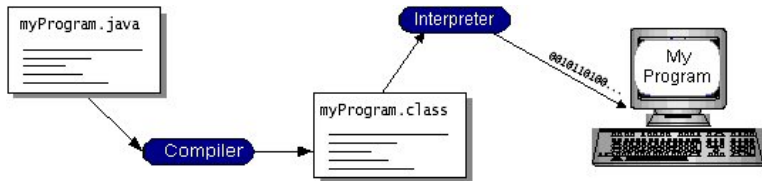
Rappels généraux

Un langage compilé/interprété

👉 **Compilation d'un programme Java : génération de byte-code**

👉 **Le byte code est :**

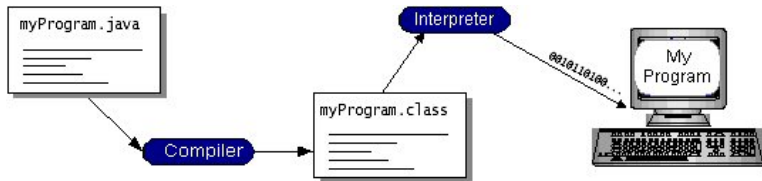
- proche d'un langage machine
- indépendant de l'environnement d'exécution (matériel + OS)



Rappels généraux

Un langage compilé/interprété

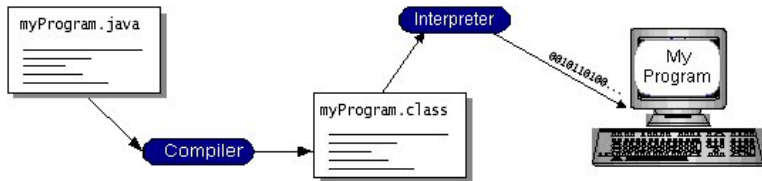
- **Compilation d'un programme Java : génération de byte-code**
- **Le byte code est :**
 - proche d'un langage machine
 - indépendant de l'environnement d'exécution (matériel + OS)



☞ Compilation d'un programme Java : génération de byte-code

☞ Le byte code est :

- proche d'un langage machine
- indépendant de l'environnement d'exécution (matériel + OS)

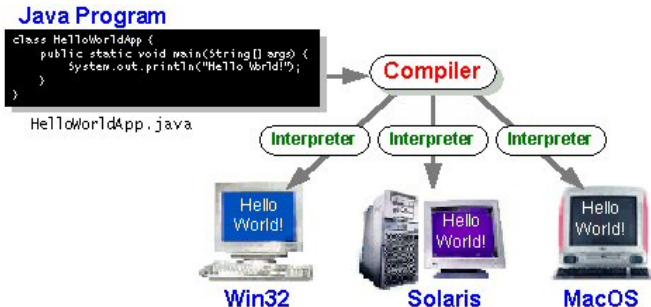


Rappels généraux

Exécution d'un programme Java compilé

☞ Le byte code assure la portabilité des programmes Java :

- langage d'une Machine Virtuelle
- à l'exécution un interpréteur simule cette machine virtuelle

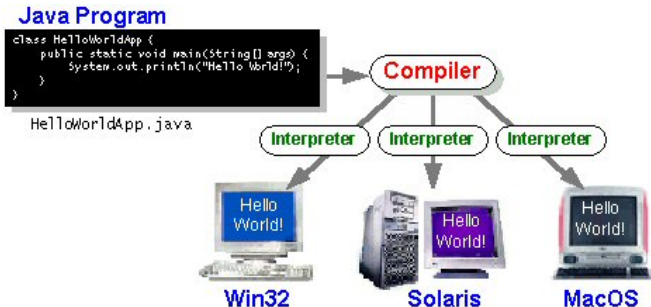


Rappels généraux

Exécution d'un programme Java compilé

☞ Le byte code assure la portabilité des programmes Java :

- langage d'une Machine Virtuelle
- à l'exécution un interpréteur simule cette machine virtuelle

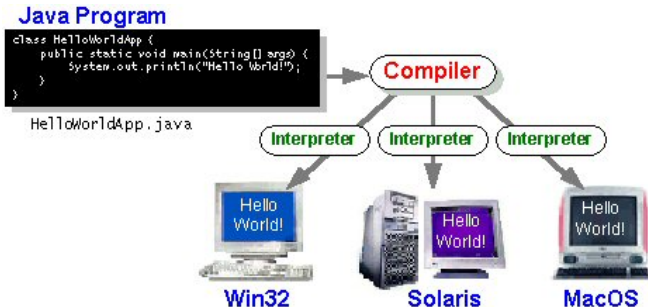


Rappels généraux

Exécution d'un programme Java compilé

☞ **Le byte code assure la portabilité des programmes Java :**

- langage d'une Machine Virtuelle
- à l'exécution un interpréteur simule cette machine virtuelle



Rappels généraux

Langages concurrents

☞ Internet C++ (et sa machine virtuelle ICVM, Open Source) :

<http://www.xmission.com/~icvm/>

☞ le C# (Microsoft C-Sharp) :

<http://www.csharpindex.com/>

```
using System;
class Welcome
{
    static void Main() {
        Console.WriteLine("Welcome to csharpindex.com");
    }
}
```

cf. comparatif

<http://www.25hoursaday.com/CsharpVsJava.html>

☞ C++, Objective-C, Smalltalk, Eiffel, perl

Rappels généraux

Langages concurrents

- ☞ Internet C++ (et sa machine virtuelle ICVM, Open Source) :

<http://www.xmission.com/~icvm/>

- ☞ le C# (Microsoft C-Sharp) :

<http://www.csharpindex.com/>

```
using System;
class Welcome
{
    static void Main() {
        Console.WriteLine("Welcome to csharpindex.com");
    }
}
```

cf. comparatif

<http://www.25hoursaday.com/CsharpVsJava.html>

- ☞ C++, Objective-C, Smalltalk, Eiffel, perl

Rappels généraux

Langages concurrents

☞ Internet C++ (et sa machine virtuelle ICVM, Open Source) :

<http://www.xmission.com/~icvm/>

☞ le C# (Microsoft C-Sharp) :

<http://www.csharpindex.com/>

```
using System;
class Welcome
{
    static void Main() {
        Console.WriteLine("Welcome to csharpindex.com");
    }
}
```

cf. comparatif

<http://www.25hoursaday.com/CsharpVsJava.html>

☞ C++, Objective-C, Smalltalk, Eiffel, perl

Rappels généraux

Notre outil de développement : BlueJ

- ☞ environnement de programmation et de développement Java (spécialement conçu pour l'enseignement) (Université de Monash, Melbourne, Australie) ;
- ☞ permet d'éditer, de compiler et d'exécuter du code Java ;
- ☞ permet de créer et d'interagir directement sur les objets (appel des méthodes associées, inspection d'objets, etc.) ;
- ☞ utilise un sous-ensemble très restreint du langage UML (Unified Modeling Language).

Rappels généraux

Notre outil de développement : BlueJ

- ☞ environnement de programmation et de développement Java (spécialement conçu pour l'enseignement) (Université de Monash, Melbourne, Australie);
- ☞ permet d'éditer, de compiler et d'exécuter du code Java;
- ☞ permet de créer et d'interagir directement sur les objets (appel des méthodes associées, inspection d'objets, etc.);
- ☞ utilise un sous-ensemble très restreint du langage UML (Unified Modeling Language).

Rappels généraux

Notre outil de développement : BlueJ

- ☞ environnement de programmation et de développement Java (spécialement conçu pour l'enseignement) (Université de Monash, Melbourne, Australie);
- ☞ permet d'éditer, de compiler et d'exécuter du code Java;
- ☞ permet de créer et d'interagir directement sur les objets (appel des méthodes associées, inspection d'objets, etc.);
- ☞ utilise un sous-ensemble très restreint du langage UML (Unified Modeling Language).

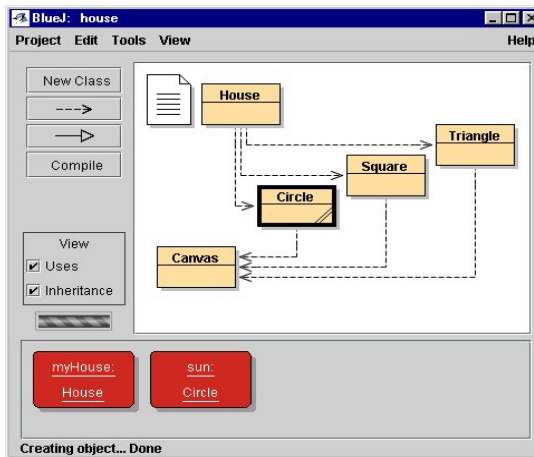
Rappels généraux

Notre outil de développement : BlueJ

- ☞ environnement de programmation et de développement Java (spécialement conçu pour l'enseignement) (Université de Monash, Melbourne, Australie);
- ☞ permet d'éditer, de compiler et d'exécuter du code Java;
- ☞ permet de créer et d'interagir directement sur les objets (appel des méthodes associées, inspection d'objets, etc.);
- ☞ utilise un sous-ensemble très restreint du langage UML (Unified Modeling Language).

Rappels généraux

L'environnement BlueJ



Classes et Objets

Les notions importantes de la programmation orientée objet

- ☞ Les structures de contrôle (idem C)
(**if** , **else** , **switch** , **for** , **do** , **while**) supposées connues.
 - ☞ Retour sur les mécanismes de déclaration de :
 - classes ;
 - attributs ;
 - méthodes.
 - ☞ Retour sur les notions de :
 - constructeur ;
 - surcharge de constructeur ;
 - surcharge de méthodes.
 - ☞ Insister sur les notions de :
 - héritage ;
 - classes abstraites ;
 - d'interfaces.
- ◇ un fil conducteur : l'exemple des coordonnées d'un point (cartésiennes, polaires, GPS)

Classes et Objets

Les notions importantes de la programmation orientée objet

- ☞ Les structures de contrôle (idem C)
(**if** , **else** , **switch** , **for** , **do** , **while**) supposées connues.
 - ☞ Retour sur les mécanismes de déclaration de :
 - **classes** ;
 - **attributs** ;
 - **méthodes**.
 - ☞ Retour sur les notions de :
 - constructeur ;
 - surcharge de constructeur ;
 - surcharge de méthodes.
 - ☞ Insister sur les notions de :
 - héritage ;
 - classes abstraites ;
 - d'interfaces.
- ◇ un fil conducteur : l'exemple des coordonnées d'un point (cartésiennes, polaires, GPS)

Classes et Objets

Les notions importantes de la programmation orientée objet

- ☞ Les structures de contrôle (idem C)
(**if** , **else** , **switch** , **for** , **do** , **while**) supposées connues.
 - ☞ Retour sur les mécanismes de déclaration de :
 - **classes** ;
 - **attributs** ;
 - **méthodes**.
 - ☞ Retour sur les notions de :
 - constructeur ;
 - surcharge de constructeur ;
 - surcharge de méthodes.
 - ☞ Insister sur les notions de :
 - héritage ;
 - classes abstraites ;
 - d'interfaces.
- ◇ un fil conducteur : l'exemple des coordonnées d'un point (cartésiennes, polaires, GPS)

Classes et Objets

Les notions importantes de la programmation orientée objet

- ☞ Les structures de contrôle (idem C)
(**if** , **else** , **switch** , **for** , **do** , **while**) supposées connues.
 - ☞ Retour sur les mécanismes de déclaration de :
 - **classes** ;
 - **attributs** ;
 - **méthodes**.
 - ☞ Retour sur les notions de :
 - constructeur ;
 - surcharge de constructeur ;
 - surcharge de méthodes.
 - ☞ Insister sur les notions de :
 - héritage ;
 - classes abstraites ;
 - d'interfaces.
- ◇ un fil conducteur : l'exemple des coordonnées d'un point
(cartésiennes, polaires, GPS)

Classes et Objets

Les notions importantes de la programmation orientée objet

- ☞ Les structures de contrôle (idem C)
(**if** , **else** , **switch** , **for** , **do** , **while**) supposées connues.
- ☞ Retour sur les mécanismes de déclaration de :
 - **classes** ;
 - **attributs** ;
 - **méthodes**.
- ☞ Retour sur les notions de :
 - constructeur ;
 - surcharge de constructeur ;
 - surcharge de méthodes.
- ☞ Insister sur les notions de :
 - héritage ;
 - classes abstraites ;
 - d'interfaces.
- ◇ un fil conducteur : l'exemple des coordonnées d'un point
(cartésiennes, polaires, GPS)

Classes et Objets

Les notions importantes de la programmation orientée objet

- ☞ Les structures de contrôle (idem C)
(**if** , **else** , **switch** , **for** , **do** , **while**) supposées connues.
 - ☞ Retour sur les mécanismes de déclaration de :
 - **classes** ;
 - **attributs** ;
 - **méthodes**.
 - ☞ Retour sur les notions de :
 - constructeur ;
 - surcharge de constructeur ;
 - surcharge de méthodes.
 - ☞ Insister sur les notions de :
 - héritage ;
 - classes abstraites ;
 - d'interfaces.
- ◇ un fil conducteur : l'exemple des coordonnées d'un point
(cartésiennes, polaires, GPS)

Classes et Objets

Les notions importantes de la programmation orientée objet

- ☞ Les structures de contrôle (idem C)
(**if** , **else** , **switch** , **for** , **do** , **while**) supposées connues.
 - ☞ Retour sur les mécanismes de déclaration de :
 - **classes** ;
 - **attributs** ;
 - **méthodes**.
 - ☞ Retour sur les notions de :
 - constructeur ;
 - surcharge de constructeur ;
 - surcharge de méthodes.
 - ☞ Insister sur les notions de :
 - héritage ;
 - classes abstraites ;
 - d'interfaces.
- ◇ un fil conducteur : l'exemple des coordonnées d'un point
(cartésiennes, polaires, GPS)

Classes et Objets

Les notions importantes de la programmation orientée objet

- ☞ Les structures de contrôle (idem C)
(**if** , **else** , **switch** , **for** , **do** , **while**) supposées connues.
 - ☞ Retour sur les mécanismes de déclaration de :
 - **classes** ;
 - **attributs** ;
 - **méthodes**.
 - ☞ Retour sur les notions de :
 - constructeur ;
 - surcharge de constructeur ;
 - surcharge de méthodes.
 - ☞ Insister sur les notions de :
 - héritage ;
 - classes abstraites ;
 - d'interfaces.
- ◇ un fil conducteur : l'exemple des coordonnées d'un point
(cartésiennes, polaires, GPS)

Classes et Objets

Les notions importantes de la programmation orientée objet

- ☞ Les structures de contrôle (idem C)
(**if** , **else** , **switch** , **for** , **do** , **while**) supposées connues.
- ☞ Retour sur les mécanismes de déclaration de :
 - **classes** ;
 - **attributs** ;
 - **méthodes**.
- ☞ Retour sur les notions de :
 - constructeur ;
 - surcharge de constructeur ;
 - surcharge de méthodes.
- ☞ Insister sur les notions de :
 - héritage ;
 - classes abstraites ;
 - d'interfaces.
- ◇ un fil conducteur : l'exemple des coordonnées d'un point
(cartésiennes, polaires, GPS)

Classes et Objets

Les notions importantes de la programmation orientée objet

- ☞ Les structures de contrôle (idem C)
(**if** , **else** , **switch** , **for** , **do** , **while**) supposées connues.
 - ☞ Retour sur les mécanismes de déclaration de :
 - **classes** ;
 - **attributs** ;
 - **méthodes**.
 - ☞ Retour sur les notions de :
 - constructeur ;
 - surcharge de constructeur ;
 - surcharge de méthodes.
 - ☞ Insister sur les notions de :
 - héritage ;
 - classes abstraites ;
 - d'interfaces.
- ◇ un fil conducteur : l'exemple des coordonnées d'un point (cartésiennes, polaires, GPS)

Classes et Objets

Les notions importantes de la programmation orientée objet

- ☞ Les structures de contrôle (idem C)
(**if** , **else** , **switch** , **for** , **do** , **while**) supposées connues.
 - ☞ Retour sur les mécanismes de déclaration de :
 - **classes** ;
 - **attributs** ;
 - **méthodes**.
 - ☞ Retour sur les notions de :
 - constructeur ;
 - surcharge de constructeur ;
 - surcharge de méthodes.
 - ☞ Insister sur les notions de :
 - héritage ;
 - classes abstraites ;
 - d'interfaces.
- ◇ un fil conducteur : l'exemple des coordonnées d'un point (cartésiennes, polaires, GPS)

Classes et Objets

Les notions importantes de la programmation orientée objet

- ☞ Les structures de contrôle (idem C)
(**if** , **else** , **switch** , **for** , **do** , **while**) supposées connues.
 - ☞ Retour sur les mécanismes de déclaration de :
 - **classes** ;
 - **attributs** ;
 - **méthodes**.
 - ☞ Retour sur les notions de :
 - constructeur ;
 - surcharge de constructeur ;
 - surcharge de méthodes.
 - ☞ Insister sur les notions de :
 - héritage ;
 - classes abstraites ;
 - d'interfaces.
- ◇ un fil conducteur : l'exemple des coordonnées d'un point (cartésiennes, polaires, GPS)

Classes et Objets

Les notions importantes de la programmation orientée objet

- ☞ Les structures de contrôle (idem C)
(**if** , **else** , **switch** , **for** , **do** , **while**) supposées connues.
 - ☞ Retour sur les mécanismes de déclaration de :
 - **classes** ;
 - **attributs** ;
 - **méthodes**.
 - ☞ Retour sur les notions de :
 - constructeur ;
 - surcharge de constructeur ;
 - surcharge de méthodes.
 - ☞ Insister sur les notions de :
 - héritage ;
 - classes abstraites ;
 - d'interfaces.
- ◇ un fil conducteur : l'exemple des coordonnées d'un point (cartésiennes, polaires, GPS)

Classes et Objets

Les notions importantes de la programmation orientée objet

- ☞ Les structures de contrôle (idem C)
(**if** , **else** , **switch** , **for** , **do** , **while**) supposées connues.
 - ☞ Retour sur les mécanismes de déclaration de :
 - **classes** ;
 - **attributs** ;
 - **méthodes**.
 - ☞ Retour sur les notions de :
 - constructeur ;
 - surcharge de constructeur ;
 - surcharge de méthodes.
 - ☞ Insister sur les notions de :
 - héritage ;
 - classes abstraites ;
 - d'interfaces.
- ◇ un fil conducteur : l'exemple des coordonnées d'un point (cartésiennes, polaires, GPS)

Des coordonnées ...

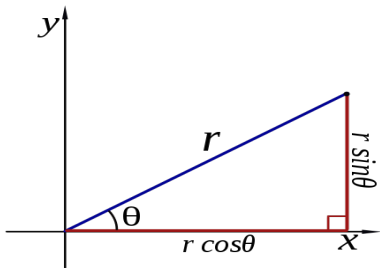


FIGURE: Coordonnées cartésiennes et polaires

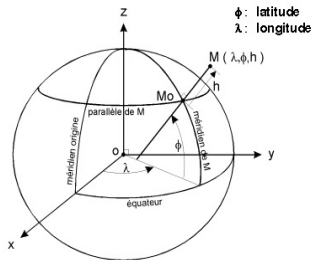


FIGURE: Coordonnées GPS

- vers la simulation logicielle d'un système de positionnement d'un point
 - ➔ positionne un point dans un repère (cartésien, polaire, GPS)
- ➔ notions d'héritage (de classe et d'interface), d'abstraction et de factorisation ;
- à partir du système de positionnement créer un GPS et construire son interface graphique ;
- ➔ structure d'une interface graphique en Java (le modèle MVC).

- vers la simulation logicielle d'un système de positionnement d'un point
 - ➔ positionne un point dans un repère (cartésien, polaire, GPS)
- ➔ notions d'héritage (de classe et d'interface), d'abstraction et de factorisation ;
- à partir du système de positionnement créer un GPS et construire son interface graphique ;
 - ➔ structure d'une interface graphique en Java (le modèle MVC).

- vers la simulation logicielle d'un système de positionnement d'un point
 - positionne un point dans un repère (cartésien, polaire, GPS)
- notions d'héritage (de classe et d'interface), d'abstraction et de factorisation ;
- à partir du système de positionnement créer un GPS et construire son interface graphique ;
 - structure d'une interface graphique en Java (le modèle MVC).

- vers la simulation logicielle d'un système de positionnement d'un point
 - positionne un point dans un repère (cartésien, polaire, GPS)
- notions d'héritage (de classe et d'interface), d'abstraction et de factorisation ;
- à partir du système de positionnement créer un GPS et construire son interface graphique ;
 - structure d'une interface graphique en Java (le modèle MVC).

- vers la simulation logicielle d'un système de positionnement d'un point
 - positionne un point dans un repère (cartésien, polaire, GPS)
- notions d'héritage (de classe et d'interface), d'abstraction et de factorisation ;
- à partir du système de positionnement créer un GPS et construire son interface graphique ;
- structure d'une interface graphique en Java (le modèle MVC).

- ☞ Une classe est constituée de :
 - *données* : que l'on nomme **attributs**
 - *procédures* : que l'on nomme **méthodes**
- ☞ Une classe est un **modèle** de définition pour des objets :
 - ayant même structure (même ensemble d'attributs) ;
 - ayant même comportement (même méthodes) ;
 - ayant une sémantique commune.
- ☞ Les **objets** sont des représentations **dynamiques** (instanciation), « vivantes » du modèle défini pour eux par la classe.
 - une classe permet d'instancier (créer) plusieurs objets ;
 - chaque objet est une *instance* d'une (seule) classe.

- ☞ Une classe est constituée de :
 - *données* : que l'on nomme **attributs**
 - *procédures* : que l'on nomme **méthodes**
- ☞ Une classe est un **modèle** de définition pour des objets :
 - ayant même structure (même ensemble d'attributs) ;
 - ayant même comportement (même méthodes) ;
 - ayant une sémantique commune.
- ☞ Les **objets** sont des représentations **dynamiques** (instanciation), « vivantes » du modèle défini pour eux par la classe.
 - une classe permet d'instancier (créer) plusieurs objets ;
 - chaque objet est une instance d'une (seule) classe.

- ☞ Une classe est constituée de :
 - *données* : que l'on nomme **attributs**
 - *procédures* : que l'on nomme **méthodes**
- ☞ Une classe est un **modèle** de définition pour des objets :
 - ayant même structure (même ensemble d'attributs) ;
 - ayant même comportement (même méthodes) ;
 - ayant une sémantique commune.
- ☞ Les **objets** sont des représentations **dynamiques** (instanciation), « vivantes » du modèle défini pour eux par la classe.
 - une classe permet d'instancier (créer) plusieurs objets ;
 - chaque objet est une instance d'une (seule) classe.

- ☞ Une classe est constituée de :
 - *données* : que l'on nomme **attributs**
 - *procédures* : que l'on nomme **méthodes**
- ☞ Une classe est un **modèle** de définition pour des objets :
 - ayant même structure (même ensemble d'attributs) ;
 - ayant même comportement (même méthodes) ;
 - ayant une sémantique commune.
- ☞ Les **objets** sont des représentations **dynamiques** (instanciation), « vivantes » du modèle défini pour eux par la classe.
 - une classe permet d'instancier (créer) plusieurs objets ;
 - chaque objet est une instance d'une (seule) classe.

- ☞ Une classe est constituée de :
 - *données* : que l'on nomme **attributs**
 - *procédures* : que l'on nomme **méthodes**
- ☞ Une classe est un **modèle** de définition pour des objets :
 - ayant même structure (même ensemble d'attributs);
 - ayant même comportement (même méthodes);
 - ayant une sémantique commune.
- ☞ Les **objets** sont des représentations **dynamiques** (instanciation), « vivantes » du modèle défini pour eux par la classe.
 - une classe permet d'instancier (créer) plusieurs objets;
 - chaque objet est une instance d'une (seule) classe.

- ☞ Une classe est constituée de :
 - *données* : que l'on nomme **attributs**
 - *procédures* : que l'on nomme **méthodes**
- ☞ Une classe est un **modèle** de définition pour des objets :
 - ayant même structure (même ensemble d'attributs);
 - ayant même comportement (même méthodes);
 - ayant une sémantique commune.
- ☞ Les **objets** sont des représentations **dynamiques** (instanciation), « vivantes » du modèle défini pour eux par la classe.
 - une classe permet d'instancier (créer) plusieurs objets;
 - chaque objet est une instance d'une (seule) classe.

- ☞ Une classe est constituée de :
 - *données* : que l'on nomme **attributs**
 - *procédures* : que l'on nomme **méthodes**
- ☞ Une classe est un **modèle** de définition pour des objets :
 - ayant même structure (même ensemble d'attributs) ;
 - ayant même comportement (même méthodes) ;
 - ayant une sémantique commune.
- ☞ Les **objets** sont des représentations **dynamiques** (instanciation), « vivantes » du modèle défini pour eux par la classe.
 - une classe permet d'instancier (créer) plusieurs objets ;
 - chaque objet est une instance d'une (seule) classe.

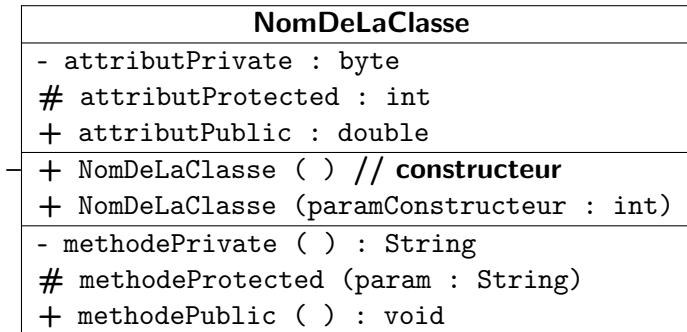
- ☞ Une classe est constituée de :
 - *données* : que l'on nomme **attributs**
 - *procédures* : que l'on nomme **méthodes**
- ☞ Une classe est un **modèle** de définition pour des objets :
 - ayant même structure (même ensemble d'attributs);
 - ayant même comportement (même méthodes);
 - ayant une sémantique commune.
- ☞ Les **objets** sont des représentations **dynamiques** (instanciation), « vivantes » du modèle défini pour eux par la classe.
 - une classe permet d'instancier (créer) plusieurs objets;
 - chaque objet est une **instance** d'une (*seule*) classe.

- ☞ Une classe est constituée de :
 - *données* : que l'on nomme **attributs**
 - *procédures* : que l'on nomme **méthodes**
- ☞ Une classe est un **modèle** de définition pour des objets :
 - ayant même structure (même ensemble d'attributs) ;
 - ayant même comportement (même méthodes) ;
 - ayant une sémantique commune.
- ☞ Les **objets** sont des représentations **dynamiques** (instanciation), « vivantes » du modèle défini pour eux par la classe.
 - une classe permet d'instancier (créer) plusieurs objets ;
 - chaque objet est une **instance** d'une (*seule*) classe.

- ☞ Une classe est constituée de :
 - *données* : que l'on nomme **attributs**
 - *procédures* : que l'on nomme **méthodes**
- ☞ Une classe est un **modèle** de définition pour des objets :
 - ayant même structure (même ensemble d'attributs) ;
 - ayant même comportement (même méthodes) ;
 - ayant une sémantique commune.
- ☞ Les **objets** sont des représentations **dynamiques** (instanciation), « vivantes » du modèle défini pour eux par la classe.
 - une classe permet d'instancier (créer) plusieurs objets ;
 - chaque objet est une **instance** d'une (**seule**) classe.

Classes et Objets

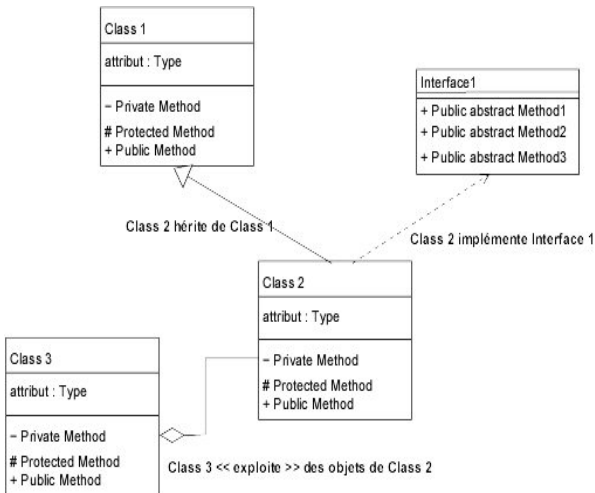
Classe : notation UML (1)



Rmq : le(s) constructeur(s) de la classe a(ont) une visibilité *public*!

Classes et Objets

Classe : notation UML (2)



Attributs

PointCartesien
- X : double
- Y : double
plus tard ici des méthodes

- **Objectif** : définir l'état de l'objet
 - attributs ou valeur des variables de l'objet (ici X, Y) à un moment particulier
 - de type **primitif** ou de type **Object**

Attributs

PointCartesien
- X : double
- Y : double
plus tard ici des méthodes

- **Objectif** : définir l'état de l'objet
 - attributs ou valeur des variables de l'objet (ici X, Y) à un moment particulier
 - de type **primitif** ou de type **Object**

Attributs

PointCartesien
- X : double
- Y : double
plus tard ici des méthodes

- **Objectif** : définir l'état de l'objet
 - attributs ou valeur des variables de l'objet (ici X,Y) à un moment particulier
 - de type **primitif** ou de type **Object**

Classes et Objets

Les types primitifs (1/2)

Syntaxe	Type	Valeur codée sur
byte	entier	8 bits, de -2^7 à $2^7 - 1$
short	entier	16 bits, de -2^{15} à $2^{15} - 1$
int	entier	32 bits, de -2^{31} à $2^{31} - 1$
long	entier	64 bits, de -2^{63} à $2^{63} - 1$
float	réel simple précision	32 bits, norme IEEE 754-1985
double	réel double précision	64 bits, norme IEEE 754-1985

TABLE: Les types primitifs numériques

La norme IEEE 754-1985 correspond à des valeurs de la forme $\pm m^2^e$, dans laquelle :

- m est un entier positif inférieur à 2^{24} et e est un entier de -149 à 104 , pour le type float ;
- m est un entier positif inférieur à 2^{53} et e est un entier de -1045 à 1000 , pour le type double

Le type primitif char

- correspond à des caractères codés en Unicode sur 16 bits. L'Unicode constitue un sur-ensemble du code ASCII.
- peut être considéré comme un entier non négatif de 0 à $65535 = 2^{16} - 1$
- Rappel : ASCII (7 bits) ; ISO Latin-1 (8 bits) ; pour coder valeur Unicode qui n'est pas ISO Latin-1 \Rightarrow `\u06f1` \Leftrightarrow chiffre 1 pour le sous-ensemble arabe oriental d'Unicode.

Le type primitif boolean

- rassemble les deux constantes logiques `true` et `false` ;
- type indépendant des types numériques. Plus possible, comme en C ou C++, d'exprimer l'équivalence « zéro \Leftrightarrow `false` » et « entier positif \Leftrightarrow `true` ».

Le type primitif char

- correspond à des caractères codés en Unicode sur 16 bits. L'Unicode constitue un sur-ensemble du code ASCII.
- peut être considéré comme un entier non négatif de 0 à $65535 = 2^{16} - 1$
- Rappel : ASCII (7 bits) ; ISO Latin-1 (8 bits) ; pour coder valeur Unicode qui n'est pas ISO Latin-1 \Rightarrow `\u06f1` \Leftrightarrow chiffre 1 pour le sous-ensemble arabe oriental d'Unicode.

Le type primitif boolean

- rassemble les deux constantes logiques `true` et `false` ;
- type indépendant des types numériques. Plus possible, comme en C ou C++, d'exprimer l'équivalence « zéro \Leftrightarrow `false` » et « entier positif \Leftrightarrow `true` ».

Le type primitif char

- correspond à des caractères codés en Unicode sur 16 bits. L'Unicode constitue un sur-ensemble du code ASCII.
- peut être considéré comme un entier non négatif de 0 à $65535 = 2^{16} - 1$
- Rappel : ASCII (7 bits) ; ISO Latin-1 (8 bits) ; pour coder valeur Unicode qui n'est pas ISO Latin-1 \Rightarrow `\u06f1` \Leftrightarrow chiffre 1 pour le sous-ensemble arabe oriental d'Unicode.

Le type primitif boolean

- rassemble les deux constantes logiques `true` et `false` ;
- type indépendant des types numériques. Plus possible, comme en C ou C++, d'exprimer l'équivalence « zéro \Leftrightarrow `false` » et « entier positif \Leftrightarrow `true` ».

Le type primitif char

- correspond à des caractères codés en Unicode sur 16 bits. L'Unicode constitue un sur-ensemble du code ASCII.
- peut être considéré comme un entier non négatif de 0 à $65535 = 2^{16} - 1$
- Rappel : ASCII (7 bits) ; ISO Latin-1 (8 bits) ; pour coder valeur Unicode qui n'est pas ISO Latin-1 \Rightarrow `\u06f1` \Leftrightarrow chiffre 1 pour le sous-ensemble arabe oriental d'Unicode.

Le type primitif boolean

- rassemble les deux constantes logiques `true` et `false` ;
- type indépendant des types numériques. Plus possible, comme en C ou C++, d'exprimer l'équivalence « zéro \Leftrightarrow `false` » et « entier positif \Leftrightarrow `true` ».

Le type primitif `char`

- correspond à des caractères codés en Unicode sur 16 bits. L'Unicode constitue un sur-ensemble du code ASCII.
- peut être considéré comme un entier non négatif de 0 à $65535 = 2^{16} - 1$
- Rappel : ASCII (7 bits) ; ISO Latin-1 (8 bits) ; pour coder valeur Unicode qui n'est pas ISO Latin-1 \Rightarrow `\u06f1` \Leftrightarrow chiffre 1 pour le sous-ensemble arabe oriental d'Unicode.

Le type primitif `boolean`

- rassemble les deux constantes logiques `true` et `false` ;
- type indépendant des types numériques. Plus possible, comme en C ou C++, d'exprimer l'équivalence `< zéro \Leftrightarrow false >` et `< entier positif \Leftrightarrow true >`.

Le type primitif `char`

- correspond à des caractères codés en Unicode sur 16 bits. L'Unicode constitue un sur-ensemble du code ASCII.
- peut être considéré comme un entier non négatif de 0 à $65535 = 2^{16} - 1$
- Rappel : ASCII (7 bits) ; ISO Latin-1 (8 bits) ; pour coder valeur Unicode qui n'est pas ISO Latin-1 \Rightarrow `\u06f1` \Leftrightarrow chiffre 1 pour le sous-ensemble arabe oriental d'Unicode.

Le type primitif `boolean`

- rassemble les deux constantes logiques `true` et `false` ;
- type indépendant des types numériques. Plus possible, comme en C ou C++, d'exprimer l'équivalence `< zéro \Leftrightarrow false >` et `< entier positif \Leftrightarrow true >`.

Le type primitif `char`

- correspond à des caractères codés en Unicode sur 16 bits. L'Unicode constitue un sur-ensemble du code ASCII.
- peut être considéré comme un entier non négatif de 0 à $65535 = 2^{16} - 1$
- Rappel : ASCII (7 bits) ; ISO Latin-1 (8 bits) ; pour coder valeur Unicode qui n'est pas ISO Latin-1 \Rightarrow `\u06f1` \Leftrightarrow chiffre 1 pour le sous-ensemble arabe oriental d'Unicode.

Le type primitif `boolean`

- rassemble les deux constantes logiques `true` et `false` ;
- type indépendant des types numériques. Plus possible, comme en C ou C++, d'exprimer l'équivalence « zéro \Leftrightarrow `false` » et « entier positif \Leftrightarrow `true` ».

Attributs : Codage en Java

```
/* Classe PointCartesien
 * author IUT 10 GEII
 * version 2010-2011 */

//LA DECLARATION D'UNE CLASSE EN JAVA SE FAIT PAR LE MOT-CLE class
public class PointCartesien
{
//DEBUT DU BLOC DE DECLARATION DE LA CLASSE

//DECLARATIONS DES ATTRIBUTS DE LA CLASSE CompteurSimple
//rmq : faire debuter le commentaire par /** permet de generer
//ensuite une doc HTML grace aux outils adequats

/** X,Y sont codes par des reels et initialises a zero */
private double X = 0;
private double Y = 0;

// ...

} //FIN DU BLOC DE DECLARATION DE LA CLASSE
```

Méthodes

- **Objectif** fournir des méthodes qui permettent
 - de connaître l'état d'un objet
 - de modifier l'état d'un objet

PointCartesien
- X : double - Y : double
pour l'instant on passe cette partie
+ getX () : double + getY () : double + afficher ()

Rmq : On trouve fréquemment des accesseurs préfixés par *set* (*get*). Ces méthodes appelées (*seteur/geteur*) sont chargées de modifier ou d'informer de l'état d'un attribut de l'objet instancié à partir de la classe.

Méthodes

- **Objectif** fournir des méthodes qui permettent
 - de connaître l'état d'un objet
 - de modifier l'état d'un objet

PointCartesien
- X : double - Y : double
pour l'instant on passe cette partie
+ getX () : double + getY () : double + afficher ()

Rmq : On trouve fréquemment des accesseurs préfixés par *set* (*get*). Ces méthodes appelées (*seteur/geteur*) sont chargées de modifier ou d'informer de l'état d'un attribut de l'objet instancié à partir de la classe.

Méthodes

- **Objectif** fournir des méthodes qui permettent
 - de connaître l'état d'un objet
 - de modifier l'état d'un objet

PointCartesien
- X : double - Y : double
pour l'instant on passe cette partie
+ getX () : double + getY () : double + afficher ()

Rmq : On trouve fréquemment des accesseurs préfixés par *set* (*get*). Ces méthodes appelées (*seteur/geteur*) sont chargées de modifier ou d'informer de l'état d'un attribut de l'objet instancié à partir de la classe.

Méthodes : Codage en Java

```
public class PointCartesien {  
  
    //DECLARATIONS DES ATTRIBUTS DE LA CLASSE PointCartesien  
    //mq : faire debuter le commentaire par /** permet de generer  
    //ensuite une doc HTML grace aux outils adequats  
  
    /** X,Y sont codes par un reel et initialises a zero */  
    private double X = 0;  
    private double Y = 0;  
  
    //DECLARATIONS DES METHODES DE LA CLASSE PointCartesien  
    //cette methode est ce que l'on nomme un accesseur  
    //elle permet de connaitre l'etat courant du point  
    /** return les coordonnees du point */  
    public double getX() {  
        return this.X;  
    }  
  
    /** afficher X */  
    public void afficher () {  
        System.out.println("X : "+this.X);  
    }  
}
```

Constructeurs

Objectif initialiser les attributs

- complètement
- avec des valeurs par défaut

PointCartesien
- X : double
- Y : double
+ PointCartesien ()
+ PointCartesien(X : double, Y : double)
+ getX () : double
+ getY () : double
+ afficher ()

Rappel : le(s) constructeur(s) de la classe a(ont) une visibilité *public*!

Rmq : Lorsque le paramètre d'un accesseur porte le nom de l'attribut de l'objet qu'il est chargé de modifier, l'auto-référence `this` permet de lever l'ambiguïté sur un identificateur (conseillé d'en faire un usage systématique : question de lisibilité du code).

Constructeurs

Objectif initialiser les attributs

- complètement
- avec des valeurs par défaut

PointCartesien
- X : double
- Y : double
+ PointCartesien ()
+ PointCartesien(X : double, Y : double)
+ getX () : double
+ getY () : double
+ afficher ()

Rappel : le(s) constructeur(s) de la classe a(ont) une visibilité *public*!

Rmq : Lorsque le paramètre d'un accesseur porte le nom de l'attribut de l'objet qu'il est chargé de modifier, l'auto-référence `this` permet de lever l'ambiguïté sur un identificateur (conseillé d'en faire un usage systématique : question de lisibilité du code).

Constructeurs

Objectif initialiser les attributs

- complètement
- avec des valeurs par défaut

PointCartesien
- X : double
- Y : double
+ PointCartesien ()
+ PointCartesien(X : double, Y : double)
+ getX () : double
+ getY () : double
+ afficher ()

Rappel : le(s) constructeur(s) de la classe a(ont) une visibilité *public*!

Rmq : Lorsque le paramètre d'un accesseur porte le nom de l'attribut de l'objet qu'il est chargé de modifier, l'auto-référence `this` permet de lever l'ambiguïté sur un identificateur (conseillé d'en faire un usage systématique : question de lisibilité du code).

Constructeurs : Codage en Java

```
public class PointCartesien {
    //DECLARATIONS DES ATTRIBUTS DE LA CLASSE PointCartesien
    //mq : faire debuter le commentaire par /** permet de generer
    //ensuite une doc HTML grace aux outils adequats
    /** etat est code par un entier et initialise a zero */
    private double X;
    private double Y;

    //DECLARATIONS DES CONSTRUCTEURS DE LA CLASSE PointCartesien
    /** Constructeur de PointCartesien sans parametre
     * X,Y initiaux du point par default a zero
     */
    public PointCartesien () {
        this.X = 0.;
        this.Y = 0.;
    }
    /** Constructeur de PointCartesien avec un parametre
     * param valX X initial du point
     */
    public PointCartesien (double X, double Y) {
        this.X = X;
        this.Y = Y;
    }
    // ...
}
```

Exercice 1

Enoncé

1. Compléter votre *PointCartesien* en prenant en compte l'attribut Y
2. Ajouter à votre *PointCartesien* les méthodes suivantes *resetX ()*, *resetY ()* et *setX (),setY ()*. Donner la nouvelle représentation UML du *PointCartesien*.
2. Concevoir un *PointPolaire* en vous inspirant de *PointCartesien* (ro en mètre et theta en degré décimal). Donner sa représentation UML.
3. Ajouter les méthodes dans *PointCartesien* et *PointPolaire* qui permettent de passer les coordonnées cartésiennes en polaires et vice et versa. Donner les nouvelles représentations UML.
4. Discuter de la conception de la classe PointPolaire si theta est en radian ou en degré sexagésimal.

Exercice 1

Enoncé

1. Compléter votre *PointCartesien* en prenant en compte l'attribut Y
2. Ajouter à votre *PointCartesien* les méthodes suivantes *resetX ()*, *resetY ()* et *setX (),setY ()*. Donner la nouvelle représentation UML du *PointCartesien*.
2. Concevoir un *PointPolaire* en vous inspirant de *PointCartesien* (ro en mètre et theta en degré décimal). Donner sa représentation UML.
3. Ajouter les méthodes dans *PointCartesien* et *PointPolaire* qui permettent de passer les coordonnées cartésiennes en polaires et vice et versa. Donner les nouvelles représentations UML.
4. Discuter de la conception de la classe PointPolaire si theta est en radian ou en degré sexagésimal.

Exercice 1

Enoncé

1. Compléter votre *PointCartesien* en prenant en compte l'attribut Y
2. Ajouter à votre *PointCartesien* les méthodes suivantes *resetX ()*, *resetY ()* et *setX (),setY ()*. Donner la nouvelle représentation UML du *PointCartesien*.
2. Concevoir un *PointPolaire* en vous inspirant de *PointCartesien* (ro en mètre et theta en degré décimal). Donner sa représentation UML.
3. Ajouter les méthodes dans *PointCartesien* et *PointPolaire* qui permettent de passer les coordonnées cartésiennes en polaires et vice et versa. Donner les nouvelles représentations UML.
4. Discuter de la conception de la classe PointPolaire si theta est en radian ou en degré sexagécimal.

Exercice 1

Enoncé

1. Compléter votre *PointCartesien* en prenant en compte l'attribut Y
2. Ajouter à votre *PointCartesien* les méthodes suivantes *resetX ()*, *resetY ()* et *setX (),setY ()*. Donner la nouvelle représentation UML du *PointCartesien*.
2. Concevoir un *PointPolaire* en vous inspirant de *PointCartesien* (ro en mètre et theta en degré décimal). Donner sa représentation UML.
3. Ajouter les méthodes dans *PointCartesien* et *PointPolaire* qui permettent de passer les coordonnées cartésiennes en polaires et vice et versa. Donner les nouvelles représentations UML.
4. Discuter de la conception de la classe PointPolaire si theta est en radian ou en degré sexagésimal.

Exercice 1

Enoncé

1. Compléter votre *PointCartesien* en prenant en compte l'attribut Y
2. Ajouter à votre *PointCartesien* les méthodes suivantes *resetX ()*, *resetY ()* et *setX (),setY ()*. Donner la nouvelle représentation UML du *PointCartesien*.
2. Concevoir un *PointPolaire* en vous inspirant de *PointCartesien* (ro en mètre et theta en degré décimal). Donner sa représentation UML.
3. Ajouter les méthodes dans *PointCartesien* et *PointPolaire* qui permettent de passer les coordonnées cartésiennes en polaires et vice et versa. Donner les nouvelles représentations UML.
4. Discuter de la conception de la classe PointPolaire si theta est en radian ou en degré sexagésimal.

Les deux coordonnées polaires r et θ peuvent être converties en coordonnées cartésiennes x et y en utilisant les fonctions trigonométriques sinus et cosinus :

$$x = r \cos \theta$$

$$y = r \sin \theta$$

Deux coordonnées cartésiennes x et y peuvent être converties en coordonnée polaire r par :

$r = \sqrt{x^2 + y^2}$, (par une simple application du théorème de Pythagore).

Pour déterminer l'angle θ , nous devons distinguer deux cas :

- Pour $r = 0$, l'angle peut prendre n'importe quelle valeur réelle.
- Pour $r \neq 0$, pour obtenir une unique valeur de θ , on se restreint à l'intervalle $[0; 2\pi[$ (ou de manière équivalente $] - \pi; \pi]$).

Conversion cartésien - polaire

Pour obtenir θ dans l'intervalle $[0; 2\pi[$, on utilise les formules suivantes (arctan désigne la réciproque de la fonction tangente) :

$$\theta = \begin{cases} \arctan\left(\frac{y}{x}\right) & \text{si } x > 0 \text{ et } y \geq 0 \\ \arctan\left(\frac{y}{x}\right) + 2\pi & \text{si } x > 0 \text{ et } y < 0 \\ \arctan\left(\frac{y}{x}\right) + \pi & \text{si } x < 0 \\ \frac{\pi}{2} & \text{si } x = 0 \text{ et } y > 0 \\ \frac{3\pi}{2} & \text{si } x = 0 \text{ et } y < 0 \end{cases}$$

Pour l'obtenir dans l'intervalle $] -\pi; \pi]$, on utilise les formules :

$$\theta = \begin{cases} \arctan\left(\frac{y}{x}\right) & \text{si } x > 0 \\ \arctan\left(\frac{y}{x}\right) + \pi & \text{si } x < 0 \text{ et } y \geq 0 \\ \arctan\left(\frac{y}{x}\right) - \pi & \text{si } x < 0 \text{ et } y < 0 \\ \frac{\pi}{2} & \text{si } x = 0 \text{ et } y > 0 \\ -\frac{\pi}{2} & \text{si } x = 0 \text{ et } y < 0 \end{cases}$$

Rmq : Pour calculer la valeur de θ vous utiliserez tout d'abord la méthode de classe `atan2()` après avoir identifié à partir de quel paquetage elle est accessible (utilisation de la documentation en ligne).

Conversion degré décimal - degré sexagésimal

- ✓ $degre = partieEntiere(degreDecimal)$
 $min = partieEntiere((degreDecimal - degre) * 60)$
 $sec = partieEntiere((((degreDecimal - degre) * 60) - min) * 60)$
- ✓ $degreDecimal = degre + (min/60) + (sec/3600)$

Les mécanismes de réutilisation

Critique des solutions proposées

- ✗ Quid de l'objectif de simuler le positionnement d'un point (cartésien, polaire, latitude, longitude, degré décimal, degré sexagésimal, radian) ?
- ✗ Fastidieux de devoir toujours recréer une nouvelle classe pour affiner un modèle de point.
- ✓ Extraire les attributs communs et les services élémentaires analogues pour les regrouper sous un même nom.
- ✓ Penser en termes de hiérarchie de classes.
- ⇒ notions de factorisation, d'héritage (de classe et d'interface) et d'abstraction.

Les mécanismes de réutilisation

Critique des solutions proposées

- ✗ Quid de l'objectif de simuler le positionnement d'un point (cartésien, polaire, latitude, longitude, degré décimal, degré sexagésimal, radian) ?
- ✗ Fastidieux de devoir toujours recréer une nouvelle classe pour affiner un modèle de point.
- ✓ Extraire les attributs communs et les services élémentaires analogues pour les regrouper sous un même nom.
- ✓ Penser en termes de hiérarchie de classes.
- ⇒ notions de factorisation, d'héritage (de classe et d'interface) et d'abstraction.

Les mécanismes de réutilisation

Critique des solutions proposées

- ✗ Quid de l'objectif de simuler le positionnement d'un point (cartésien, polaire, latitude, longitude, degré décimal, degré sexagésimal, radian) ?
- ✗ Fastidieux de devoir toujours recréer une nouvelle classe pour affiner un modèle de point.
- ✓ Extraire les attributs communs et les services élémentaires analogues pour les regrouper sous un même nom.
- ✓ Penser en termes de hiérarchie de classes.
- ⇒ notions de factorisation, d'héritage (de classe et d'interface) et d'abstraction.

Les mécanismes de réutilisation

Critique des solutions proposées

- ✗ Quid de l'objectif de simuler le positionnement d'un point (cartésien, polaire, latitude, longitude, degré décimal, degré sexagésimal, radian) ?
 - ✗ Fastidieux de devoir toujours recréer une nouvelle classe pour affiner un modèle de point.
 - ✓ Extraire les attributs communs et les services élémentaires analogues pour les regrouper sous un même nom.
 - ✓ Penser en termes de hiérarchie de classes.
- ⇒ notions de factorisation, d'héritage (de classe et d'interface) et d'abstraction.

Les mécanismes de réutilisation

Critique des solutions proposées

- ✗ Quid de l'objectif de simuler le positionnement d'un point (cartésien, polaire, latitude, longitude, degré décimal, degré sexagésimal, radian) ?
- ✗ Fastidieux de devoir toujours recréer une nouvelle classe pour affiner un modèle de point.
- ✓ Extraire les attributs communs et les services élémentaires analogues pour les regrouper sous un même nom.
- ✓ Penser en termes de hiérarchie de classes.
- ➡ notions de factorisation, d'héritage (de classe et d'interface) et d'abstraction.