Application à la POO – ARIVE – 3

Christophe BLANC

IUT Montluçon
Université Blaise Pascal

Janvier 2016

Créer son propre paquetage (package)

Introduction

- Projet sur le GPS :
  - plusieurs classes *PointDegDec, PointDegSexa, ...*, et une interface.
  - *imaginer un projet plus important ...*
- ces classes peuvent être utiles pour d'autres projets
- pour s'y retrouver :
  - hiérarchiser fonctionnellement l'ensemble des classes mises à disposition du programmeur (à ne pas confondre avec la "hiérarchie" induite par l'héritage de classes ou d'interfaces).

⇨ les regrouper dans un paquetage (package)

Créer son propre paquetage (package)

Introduction

- Projet sur le GPS :
  - plusieurs classes *PointDegDec, PointDegSexa, ...*, et une interface.
  - *imaginer un projet plus important ...*
- ces classes peuvent être utiles pour d'autres projets
- pour s'y retrouver :
  - hiérarchiser fonctionnellement l'ensemble des classes mises à disposition du programmeur (à ne pas confondre avec la "hiérarchie" induite par l'héritage de classes ou d'interfaces).

⇨ les regrouper dans un paquetage (package)

Créer son propre paquetage (package)

Introduction

- Projet sur le GPS :
  - plusieurs classes *PointDegDec, PointDegSexa, ...*, et une interface.
  - *imaginer un projet plus important ...*
- ces classes peuvent être utiles pour d'autres projets
- pour s'y retrouver :
  - hiérarchiser fonctionnellement l'ensemble des classes mises à disposition du programmeur (à ne pas confondre avec la "hiérarchie" induite par l'héritage de classes ou d'interfaces).

⇨ les regrouper dans un paquetage (package)

Créer son propre paquetage (package)

Introduction

- Projet sur le GPS :
  - plusieurs classes *PointDegDec, PointDegSexa, ...*, et une interface.
  - *imaginer un projet plus important ...*
- ces classes peuvent être utiles pour d'autres projets
- pour s'y retrouver :
  - hiérarchiser fonctionnellement l'ensemble des classes mises à disposition du programmeur (à ne pas confondre avec la "hiérarchie" induite par l'héritage de classes ou d'interfaces).

⇨ les regrouper dans un paquetage (package)

Créer son propre paquetage (package)

Introduction

- Projet sur le GPS :
  - plusieurs classes *PointDegDec, PointDegSexa, ...*, et une interface.
  - *imaginer un projet plus important ...*
- ces classes peuvent être utiles pour d'autres projets
- pour s'y retrouver :
  - hiérarchiser fonctionnellement l'ensemble des classes mises à disposition du programmeur (à ne pas confondre avec la "hiérarchie" induite par l'héritage de classes ou d'interfaces).

⇨ les regrouper dans un paquetage (package)

- Projet sur le GPS :
  - plusieurs classes *PointDegDec, PointDegSexa, ...*, et une interface.
  - *imaginer un projet plus important ...*
- ces classes peuvent être utiles pour d'autres projets
- pour s'y retrouver :
  - hiérarchiser fonctionnellement l'ensemble des classes mises à disposition du programmeur (à ne pas confondre avec la "hiérarchie" induite par l'héritage de classes ou d'interfaces).

⇨ les regrouper dans un paquetage (package)

- Projet sur le GPS :
  - plusieurs classes *PointDegDec, PointDegSexa, ...*, et une interface.
  - *imaginer un projet plus important ...*
- ces classes peuvent être utiles pour d'autres projets
- pour s'y retrouver :
  - hiérarchiser fonctionnellement l'ensemble des classes mises à disposition du programmeur (à ne pas confondre avec la "hiérarchie" induite par l'héritage de classes ou d'interfaces).

⇨ les regrouper dans un paquetage (package)

- Créer son package *pointGPS*
  - un répertoire *pointGPS* est associé à ce paquetage
  - par convention, le nom des paquetages est en minuscules
  - déclarer le package en première instruction du fichier (une seule fois dans le fichier source : une classe ► un package)

```
1  // nom du site WEB a partir duquel il est distribue
2  package fr.nom-de-domaine-internet.pointGPS;
3
4  public class Point
5  {
6      // ici la definition de la classe
7  }
```

```
1  package pointGPS;  // pour une utlisation avec BlueJ (en local)
2
3  public class Point
4  {
5      // ici la definition de la classe
6  }
```

Créer son propre paquetage (package)

Créer un package

- Créer son package *pointGPS*
  - un répertoire *pointGPS* est associé à ce paquetage
  - par convention, le nom des paquetages est en minuscules
  - déclarer le package en première instruction du fichier (une seule fois dans le fichier source : une classe ► un package)

```
1  // nom du site WEB a partir duquel il est distribue
2  package fr.nom-de-domaine-internet.pointGPS;
3
4  public class Point
5  {
6      // ici la definition de la classe
7  }
```

```
1  package pointGPS;  // pour une utlisation avec BlueJ (en local)
2
3  public class Point
4  {
5      // ici la definition de la classe
6  }
```

- Créer son package *pointGPS*
  - un répertoire *pointGPS* est associé à ce paquetage
  - par convention, le nom des paquetages est en minuscules
  - déclarer le package en première instruction du fichier (une seule fois dans le fichier source : une classe ► un package)

```
1  // nom du site WEB a partir duquel il est distribue
2  package fr.nom-de-domaine-internet.pointGPS;
3
4  public class Point
5  {
6      // ici la definition de la classe
7  }
```

```
1  package pointGPS;  // pour une utlisation avec BlueJ (en local)
2
3  public class Point
4  {
5      // ici la definition de la classe
6  }
```

- Créer son package *pointGPS*
  - un répertoire *pointGPS* est associé à ce paquetage
  - par convention, le nom des paquetages est en minuscules
  - déclarer le package en première instruction du fichier (une seule fois dans le fichier source : une classe ► un package)

```
1  // nom du site WEB a partir duquel il est distribue
2  package fr.nom-de-domaine-internet.pointGPS;
3
4  public class Point
5  {
6      // ici la definition de la classe
7  }
```

```
1  package pointGPS;  // pour une utlisation avec BlueJ (en local)
2
3  public class Point
4  {
5      // ici la definition de la classe
6  }
```

- Créer son package *pointGPS*
  - un répertoire *pointGPS* est associé à ce paquetage
  - par convention, le nom des paquetages est en minuscules
  - déclarer le package en première instruction du fichier (une seule fois dans le fichier source : une classe ► un package)

```
1  // nom du site WEB a partir duquel il est distribue
2  package fr.nom-de-domaine-internet.pointGPS;
3
4  public class Point
5  {
6      // ici la definition de la classe
7  }
```

```
1  package pointGPS;  // pour une utlisation avec BlueJ (en local)
2
3  public class Point
4  {
5      // ici la definition de la classe
6  }
```

- Créer son package *pointGPS*
  - un répertoire *pointGPS* est associé à ce paquetage
  - par convention, le nom des paquetages est en minuscules
  - déclarer le package en première instruction du fichier (une seule fois dans le fichier source : une classe ► un package)

```
1  // nom du site WEB a partir duquel il est distribue
2  package fr.nom-de-domaine-internet.pointGPS;
3
4  public class Point
5  {
6      // ici la definition de la classe
7  }
```

```
1  package pointGPS;  // pour une utlisation avec BlueJ (en local)
2
3  public class Point
4  {
5      // ici la definition de la classe
6  }
```

- Utiliser le package *pointGPS*
  - importer le package dans le fichier source qui l'exploite
  - instruction **import** en en-tête du fichier source
  - les classes du **package java.lang** sont automatiquement et implicitement importées dans les fichiers sources.

```
1  import pointGPS.PointDegDec; // seule la classe PointDegDec est importee
2  import java.lang.*           // import de toutes les classes du package lang
3                               // pas necessaire car toujours fait automatiquement
4
5  public class Trajectoire  {
6      private ISvcPointGPS point;
7
8      // ici la definition de la classe
9  }
```

- Utiliser le package *pointGPS*
  - importer le package dans le fichier source qui l'exploite
  - instruction **import** en en-tête du fichier source
  - les classes du **package java.lang** sont automatiquement et implicitement importées dans les fichiers sources.

```
1  import pointGPS.PointDegDec; // seule la classe PointDegDec est importee
2  import java.lang.*                // import de toutes les classes du package lang
3                                    // pas necessaire car toujours fait automatiquement
4
5  public class Trajectoire  {
6      private ISvcPointGPS point;
7
8      // ici la definition de la classe
9  }
```

- Utiliser le package *pointGPS*
  - importer le package dans le fichier source qui l'exploite
  - instruction **import** en en-tête du fichier source
  - les classes du **package java.lang** sont automatiquement et implicitement importées dans les fichiers sources.

```
1  import pointGPS.PointDegDec; // seule la classe PointDegDec est importee
2  import java.lang.*           // import de toutes les classes du package lang
3                               // pas necessaire car toujours fait automatiquement
4
5  public class Trajectoire  {
6      private ISvcPointGPS point;
7
8      // ici la definition de la classe
9  }
```

- Utiliser le package *pointGPS*
  - importer le package dans le fichier source qui l'exploite
  - instruction **import** en en-tête du fichier source
  - les classes du **package java.lang** sont automatiquement et implicitement importées dans les fichiers sources.

```
1  import pointGPS.PointDegDec; // seule la classe PointDegDec est importee
2  import java.lang.*                // import de toutes les classes du package lang
3                                    // pas necessaire car toujours fait automatiquement
4
5  public class Trajectoire  {
6      private ISvcPointGPS point;
7
8      // ici la definition de la classe
9  }
```

- Utiliser le package *pointGPS*
  - importer le package dans le fichier source qui l'exploite
  - instruction **import** en en-tête du fichier source
  - les classes du **package java.lang** sont automatiquement et implicitement importées dans les fichiers sources.

```
1  import pointGPS.PointDegDec; // seule la classe PointDegDec est importee
2  import java.lang.*                // import de toutes les classes du package lang
3                                    // pas necessaire car toujours fait automatiquement
4
5  public class Trajectoire  {
6      private ISvcPointGPS point;
7
8      // ici la definition de la classe
9  }
```

- Utiliser le package *pointGPS*
  - importer le package dans le fichier source qui l'exploite
  - instruction **import** en en-tête du fichier source
  - les classes du **package java.lang** sont automatiquement et implicitement importées dans les fichiers sources.

```
1  import pointGPS.PointDegDec; // seule la classe PointDegDec est importee
2  import java.lang.*                // import de toutes les classes du package lang
3                                    // pas necessaire car toujours fait automatiquement
4
5  public class Trajectoire  {
6      private ISvcPointGPS point;
7
8      // ici la definition de la classe
9  }
```

- une application Java ◄► Somme de paquetages...
- un paquetage Java ◄► Somme de fichiers **.java** ...
- un fichier **.java** ◄► Somme de classes dont une seule est **public**.
  Le nom du fichier est celui de la classe public à laquelle on ajoute l'extension **.java**
- en plus d'être un langage de programmation, Java dispose d'une API (Application Programming Interface)
  riche : nombreux packages standards
- programmer en Java nécessite une bonne connaissance de ces packages.

- une application Java ◄► Somme de paquetages...
- un paquetage Java ◄► Somme de fichiers **.java** ...
- un fichier **.java** ◄► Somme de classes dont une seule est **public**.
  Le nom du fichier est celui de la classe public à laquelle on ajoute l'extension **.java**
- en plus d'être un langage de programmation, Java dispose d'une API (Application Programming Interface)
  riche : nombreux packages standards
- programmer en Java nécessite une bonne connaissance de ces packages.

- une application Java ◄► Somme de paquetages...
- un paquetage Java ◄► Somme de fichiers **.java** ...
- un fichier **.java** ◄► Somme de classes dont une seule est **public**.
  Le nom du fichier est celui de la classe public à laquelle on ajoute l'extension **.java**
- en plus d'être un langage de programmation, Java dispose d'une API (Application Programming Interface)
  riche : nombreux packages standards
- programmer en Java nécessite une bonne connaissance de ces packages.

- une application Java ◄► Somme de paquetages...
- un paquetage Java ◄► Somme de fichiers **.java** ...
- un fichier **.java** ◄► Somme de classes dont une seule est **public**.
  Le nom du fichier est celui de la classe public à laquelle on ajoute l'extension **.java**
- en plus d'être un langage de programmation, Java dispose d'une API (Application Programming Interface)
  riche : nombreux packages standards
- programmer en Java nécessite une bonne connaissance de ces packages.

- une application Java ◄► Somme de paquetages...
- un paquetage Java ◄► Somme de fichiers **.java** ...
- un fichier **.java** ◄► Somme de classes dont une seule est **public**.
  Le nom du fichier est celui de la classe public à laquelle on ajoute l'extension **.java**
- en plus d'être un langage de programmation, Java dispose d'une API (Application Programming Interface)
  riche : nombreux packages standards
- programmer en Java nécessite une bonne connaissance de ces packages.

Exploiter un paquetage (package)

Packages de classes standards

- java.lang : classes essentielles
  - objet, types de base
- java.util : structures de données (collections)
  - listes, ensembles, hashtables, arbres, itérateurs ...
- java.awt : interface graphique (Abstract Window Toolkit)
  - fenêtres, boutons, événements...
- java.io : entrées / sorties
  - flot de données issu d'un fichier, d'un buffer, d'un « pipe ».
- java.net : réseau
  - URL, sockets
- javax.swings : interface graphique
  - composants de plus haut niveau que ceux de **awt**
  - exploitation du modéle MVC (Model View Controler)
- ...

Exploiter un paquetage (package)

Packages de classes standards

- java.lang : classes essentielles
  - objet, types de base
- java.util : structures de données (collections)
  - listes, ensembles, hashtables, arbres, itérateurs ...
- java.awt : interface graphique (Abstract Window Toolkit)
  - fenêtres, boutons, événements...
- java.io : entrées / sorties
  - flot de données issu d'un fichier, d'un buffer, d'un « pipe ».
- java.net : réseau
  - URL, sockets
- javax.swings : interface graphique
  - composants de plus haut niveau que ceux de **awt**
  - exploitation du modéle MVC (Model View Controler)
- ...

Exploiter un paquetage (package)

Packages de classes standards

- java.lang : classes essentielles
  - objet, types de base
- java.util : structures de données (collections)
  - listes, ensembles, hashtables, arbres, itérateurs ...
- java.awt : interface graphique (Abstract Window Toolkit)
  - fenêtres, boutons, événements...
- java.io : entrées / sorties
  - flot de données issu d'un fichier, d'un buffer, d'un « pipe ».
- java.net : réseau
  - URL, sockets
- javax.swings : interface graphique
  - composants de plus haut niveau que ceux de **awt**
  - exploitation du modéle MVC (Model View Controler)
- ...

Exploiter un paquetage (package)

Packages de classes standards

- java.lang : classes essentielles
  - objet, types de base
- java.util : structures de données (collections)
  - listes, ensembles, hashtables, arbres, itérateurs ...
- java.awt : interface graphique (Abstract Window Toolkit)
  - fenêtres, boutons, événements...
- java.io : entrées / sorties
  - flot de données issu d'un fichier, d'un buffer, d'un « pipe ».
- java.net : réseau
  - URL, sockets
- javax.swings : interface graphique
  - composants de plus haut niveau que ceux de **awt**
  - exploitation du modéle MVC (Model View Controler)
- ...

Exploiter un paquetage (package)

Packages de classes standards

- java.lang : classes essentielles
  - objet, types de base
- java.util : structures de données (collections)
  - listes, ensembles, hashtables, arbres, itérateurs ...
- java.awt : interface graphique (Abstract Window Toolkit)
  - fenêtres, boutons, événements...
- java.io : entrées / sorties
  - flot de données issu d'un fichier, d'un buffer, d'un « pipe ».
- java.net : réseau
  - URL, sockets
- javax.swings : interface graphique
  - composants de plus haut niveau que ceux de **awt**
  - exploitation du modéle MVC (Model View Controler)
- ...

Exploiter un paquetage (package)

Packages de classes standards

- java.lang : classes essentielles
  - objet, types de base
- java.util : structures de données (collections)
  - listes, ensembles, hashtables, arbres, itérateurs ...
- java.awt : interface graphique (Abstract Window Toolkit)
  - fenêtres, boutons, événements...
- java.io : entrées / sorties
  - flot de données issu d'un fichier, d'un buffer, d'un « pipe ».
- java.net : réseau
  - URL, sockets
- javax.swings : interface graphique
  - composants de plus haut niveau que ceux de **awt**
  - exploitation du modéle MVC (Model View Controler)
- ...

Exploiter un paquetage (package)

Packages de classes standards

- java.lang : classes essentielles
  - objet, types de base
- java.util : structures de données (collections)
  - listes, ensembles, hashtables, arbres, itérateurs ...
- java.awt : interface graphique (Abstract Window Toolkit)
  - fenêtres, boutons, événements...
- java.io : entrées / sorties
  - flot de données issu d'un fichier, d'un buffer, d'un « pipe ».
- java.net : réseau
  - URL, sockets
- javax.swings : interface graphique
  - composants de plus haut niveau que ceux de **awt**
  - exploitation du modéle MVC (Model View Controler)
- ...

Exploiter un paquetage (package)

Packages de classes standards

- java.lang : classes essentielles
  - objet, types de base
- java.util : structures de données (collections)
  - listes, ensembles, hashtables, arbres, itérateurs ...
- java.awt : interface graphique (Abstract Window Toolkit)
  - fenêtres, boutons, événements...
- java.io : entrées / sorties
  - flot de données issu d'un fichier, d'un buffer, d'un « pipe ».
- java.net : réseau
  - URL, sockets
- javax.swings : interface graphique
  - composants de plus haut niveau que ceux de **awt**
  - exploitation du modéle MVC (Model View Controler)
- ...

Exploiter un paquetage (package)

Packages de classes standards

- java.lang : classes essentielles
  - objet, types de base
- java.util : structures de données (collections)
  - listes, ensembles, hashtables, arbres, itérateurs ...
- java.awt : interface graphique (Abstract Window Toolkit)
  - fenêtres, boutons, événements...
- java.io : entrées / sorties
  - flot de données issu d'un fichier, d'un buffer, d'un « pipe ».
- java.net : réseau
  - URL, sockets
- javax.swings : interface graphique
  - composants de plus haut niveau que ceux de **awt**
  - exploitation du modéle MVC (Model View Controler)
- ...

Exploiter un paquetage (package)

Packages de classes standards

- java.lang : classes essentielles
  - objet, types de base
- java.util : structures de données (collections)
  - listes, ensembles, hashtables, arbres, itérateurs ...
- java.awt : interface graphique (Abstract Window Toolkit)
  - fenêtres, boutons, événements...
- java.io : entrées / sorties
  - flot de données issu d'un fichier, d'un buffer, d'un « pipe ».
- java.net : réseau
  - URL, sockets
- javax.swings : interface graphique
  - composants de plus haut niveau que ceux de **awt**
  - exploitation du modéle MVC (Model View Controler)
- ...

Exploiter un paquetage (package)

Packages de classes standards

- java.lang : classes essentielles
  - objet, types de base
- java.util : structures de données (collections)
  - listes, ensembles, hashtables, arbres, itérateurs ...
- java.awt : interface graphique (Abstract Window Toolkit)
  - fenêtres, boutons, événements...
- java.io : entrées / sorties
  - flot de données issu d'un fichier, d'un buffer, d'un « pipe ».
- java.net : réseau
  - URL, sockets
- javax.swings : interface graphique
  - composants de plus haut niveau que ceux de **awt**
  - exploitation du modéle MVC (Model View Controler)
- ...

Exploiter un paquetage (package)

Packages de classes standards

- java.lang : classes essentielles
  - objet, types de base
- java.util : structures de données (collections)
  - listes, ensembles, hashtables, arbres, itérateurs ...
- java.awt : interface graphique (Abstract Window Toolkit)
  - fenêtres, boutons, événements...
- java.io : entrées / sorties
  - flot de données issu d'un fichier, d'un buffer, d'un « pipe ».
- java.net : réseau
  - URL, sockets
- javax.swings : interface graphique
  - composants de plus haut niveau que ceux de **awt**
  - exploitation du modéle MVC (Model View Controler)
- ...

Exploiter un paquetage (package)

Packages de classes standards

- java.lang : classes essentielles
  - objet, types de base
- java.util : structures de données (collections)
  - listes, ensembles, hashtables, arbres, itérateurs ...
- java.awt : interface graphique (Abstract Window Toolkit)
  - fenêtres, boutons, événements...
- java.io : entrées / sorties
  - flot de données issu d'un fichier, d'un buffer, d'un « pipe ».
- java.net : réseau
  - URL, sockets
- javax.swings : interface graphique
  - composants de plus haut niveau que ceux de **awt**
  - exploitation du modéle MVC (Model View Controler)
- ...

Exploiter un paquetage (package)

Packages de classes standards

- java.lang : classes essentielles
  - objet, types de base
- java.util : structures de données (collections)
  - listes, ensembles, hashtables, arbres, itérateurs ...
- java.awt : interface graphique (Abstract Window Toolkit)
  - fenêtres, boutons, événements...
- java.io : entrées / sorties
  - flot de données issu d'un fichier, d'un buffer, d'un « pipe ».
- java.net : réseau
  - URL, sockets
- javax.swings : interface graphique
  - composants de plus haut niveau que ceux de **awt**
  - exploitation du modéle MVC (Model View Controler)
- ...

Exercice 4
Enoncé

Dans l'exercice 2, nous avons créé un point GPS polymorphique (point en degré et/ou en radian)
Nous allons utiliser ce modèle pour créer une classe Trajectoire qui permet d'afficher une liste de points GPS.
Les services sont définis par l'interface.

```
1  import PointGPS.*;
2
3
4
5  public interface ISvcTrajectoire
6  {
7     public abstract void ajouterPointDeg(double latitude, double longitude);
8     public abstract void ajouterPointDeg(int[] latitude, int[] longitude);
9     public abstract void ajouterPointRad(double latitude, double longitude);
10    public abstract void enleverPoint(int index);
11    public abstract String toString();
12
13 }
```

Exercice 4
Enoncé (suite)

1. Créer un nouveau projet BlueJ et saisir l'interface *ISvcTrajectoire.java*.
2. Créer le paquetage *pointGPS*.
3. Créer la classe *Trajectoire* qui implémente *ISvcTrajectoire.java*. Un attribut de type ArrayList<*ISvcPointGPS*> sera défini.
4. Effectuer les tests (créer une classe Test) pour vous assurer du codage correct de la trajectoire.

Exercice 4
Enoncé (suite)

1. Créer un nouveau projet BlueJ et saisir l'interface *ISvcTrajectoire.java*.
2. Créer le paquetage *pointGPS*.
3. Créer la classe *Trajectoire* qui implémente *ISvcTrajectoire.java*. Un attribut de type ArrayList<*ISvcPointGPS*> sera défini.
4. Effectuer les tests (créer une classe Test) pour vous assurer du codage correct de la trajectoire.

Exercice 4
Enoncé (suite)

1. Créer un nouveau projet BlueJ et saisir l'interface *ISvcTrajectoire.java*.
2. Créer le paquetage *pointGPS*.
3. Créer la classe *Trajectoire* qui implémente *ISvcTrajectoire.java*. Un attribut de type ArrayList<*ISvcPointGPS*> sera défini.
4. Effectuer les tests (créer une classe Test) pour vous assurer du codage correct de la trajectoire.

Exercice 4
Enoncé (suite)

1. Créer un nouveau projet BlueJ et saisir l'interface *ISvcTrajectoire.java*.
2. Créer le paquetage *pointGPS*.
3. Créer la classe *Trajectoire* qui implémente *ISvcTrajectoire.java*. Un attribut de type ArrayList<*ISvcPointGPS*> sera défini.
4. Effectuer les tests (créer une classe Test) pour vous assurer du codage correct de la trajectoire.

Le polymorphisme

- ☞ Pour la classe **Trajectoire** , les références **pointDegre** et **pointRadian** de type **ISvcPointGPS** peuvent être liées à des objets de type **PointDegDec** ou **PointDegSexa** ou **PointDegRad** car ils implémentent **ISvcPointGPS** (ou héritent d'une telle classe).
- ➪ Cette capacité des références de pouvoir être liées à des types différents est appelée *polymorphisme* (litt. plusieurs formes).
- ✏ En java, le polymorphisme est possible si et seulement si les types ont été unifiés.
- ✏ Le type de la référence (**ISvcPointGPS**) est appelé le type *statique*.
- ✏ Le type de l'objet lié (**PointDegDec** ou **PointDegSexa** ou **PointDegRad**) est appelé le type *dynamique*.
- ➥ **Intérét : PointDegDec** ou **PointDegSexa** ou **PointDegRad**, ou toute classe qui en hérite, peuvent évoluer, **Trajectoire** est toujours OK.

Le polymorphisme

- ☞ Pour la classe **Trajectoire** , les références **pointDegre** et **pointRadian** de type **ISvcPointGPS** peuvent être liées à des objets de type **PointDegDec** ou **PointDegSexa** ou **PointDegRad** car ils implémentent **ISvcPointGPS** (ou héritent d'une telle classe).
- ➪ Cette capacité des références de pouvoir être liées à des types différents est appelée *polymorphisme* (litt. plusieurs formes).
- ✏ En java, le polymorphisme est possible si et seulement si les types ont été unifiés.
- ✏ Le type de la référence (**ISvcPointGPS**) est appelé le type *statique*.
- ✏ Le type de l'objet lié (**PointDegDec** ou **PointDegSexa** ou **PointDegRad**) est appelé le type *dynamique*.
- ➥ **Intérét : PointDegDec** ou **PointDegSexa** ou **PointDegRad**, ou toute classe qui en hérite, peuvent évoluer, **Trajectoire** est toujours OK.

Le polymorphisme

- ☞ Pour la classe **Trajectoire** , les références **pointDegre** et **pointRadian** de type **ISvcPointGPS** peuvent être liées à des objets de type **PointDegDec** ou **PointDegSexa** ou **PointDegRad** car ils implémentent **ISvcPointGPS** (ou héritent d'une telle classe).
- ➪ Cette capacité des références de pouvoir être liées à des types différents est appelée *polymorphisme* (litt. plusieurs formes).
- ✏ En java, le polymorphisme est possible si et seulement si les types ont été unifiés.
- ✏ Le type de la référence (**ISvcPointGPS**) est appelé le type *statique*.
- ✏ Le type de l'objet lié (**PointDegDec** ou **PointDegSexa** ou **PointDegRad**) est appelé le type *dynamique*.
- ➥ **Intérét : PointDegDec** ou **PointDegSexa** ou **PointDegRad**, ou toute classe qui en hérite, peuvent évoluer, **Trajectoire** est toujours OK.

Le polymorphisme

- ☞ Pour la classe **Trajectoire** , les références **pointDegre** et **pointRadian** de type **ISvcPointGPS** peuvent être liées à des objets de type **PointDegDec** ou **PointDegSexa** ou **PointDegRad** car ils implémentent **ISvcPointGPS** (ou héritent d'une telle classe).
- ➪ Cette capacité des références de pouvoir être liées à des types différents est appelée *polymorphisme* (litt. plusieurs formes).
- ✏ En java, le polymorphisme est possible si et seulement si les types ont été unifiés.
- ✏ Le type de la référence (**ISvcPointGPS**) est appelé le type *statique*.
- ✏ Le type de l'objet lié (**PointDegDec** ou **PointDegSexa** ou **PointDegRad**) est appelé le type *dynamique*.
- ➥ **Intérét : PointDegDec** ou **PointDegSexa** ou **PointDegRad**, ou toute classe qui en hérite, peuvent évoluer, **Trajectoire** est toujours OK.

Le polymorphisme

- ☞ Pour la classe **Trajectoire** , les références **pointDegre** et **pointRadian** de type **ISvcPointGPS** peuvent être liées à des objets de type **PointDegDec** ou **PointDegSexa** ou **PointDegRad** car ils implémentent **ISvcPointGPS** (ou héritent d'une telle classe).
- ➪ Cette capacité des références de pouvoir être liées à des types différents est appelée *polymorphisme* (litt. plusieurs formes).
- ✏ En java, le polymorphisme est possible si et seulement si les types ont été unifiés.
- ✏ Le type de la référence (**ISvcPointGPS**) est appelé le type *statique*.
- ✏ Le type de l'objet lié (**PointDegDec** ou **PointDegSexa** ou **PointDegRad**) est appelé le type *dynamique*.
- ➥ **Intérét : PointDegDec** ou **PointDegSexa** ou **PointDegRad**, ou toute classe qui en hérite, peuvent évoluer, **Trajectoire** est toujours OK.

Le polymorphisme

- ☞ Pour la classe **Trajectoire** , les références **pointDegre** et **pointRadian** de type **ISvcPointGPS** peuvent être liées à des objets de type **PointDegDec** ou **PointDegSexa** ou **PointDegRad** car ils implémentent **ISvcPointGPS** (ou héritent d'une telle classe).
- ➪ Cette capacité des références de pouvoir être liées à des types différents est appelée *polymorphisme* (litt. plusieurs formes).
- ✏ En java, le polymorphisme est possible si et seulement si les types ont été unifiés.
- ✏ Le type de la référence (**ISvcPointGPS**) est appelé le type *statique*.
- ✏ Le type de l'objet lié (**PointDegDec** ou **PointDegSexa** ou **PointDegRad**) est appelé le type *dynamique*.
- ➥ **Intérét : PointDegDec** ou **PointDegSexa** ou **PointDegRad**, ou toute classe qui en hérite, peuvent évoluer, **Trajectoire** est toujours OK.