

Application à la POO

– 2 –

Christophe BLANC

IUT D'ALLIER
UNIVERSITÉ Clermont Auvergne

Septembre 2018

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.

Les mécanismes de réutilisation

Principes d'une bonne conception "objet"

① Définir en premier les services que doit réaliser l'objet (fonctionnalités souhaitées)

- les définir avec précision ;
- programmation par "contrat" ;
- en "Java", utiliser les interfaces.

② Choisir ensuite le type (la classe) des attributs

- le plus adapté au problème à résoudre ;
- exploiter les services offerts par les classes des bibliothèques de l'outil de développement ;
- si remise en cause des choix (des types ou de l'algorithmie), respecter toujours le contrat (possible par l'utilisation des interfaces).

③ Si besoin de faire évoluer le cahier des charges

- modifier le contrat en ajoutant des interfaces ;
- implémenter ces nouvelles interfaces.

Les mécanismes de réutilisation

Principes d'une bonne conception "objet"

- 1 **Définir en premier les services que doit réaliser l'objet (fonctionnalités souhaitées)**
 - les définir avec précision ;
 - programmation par "contrat" ;
 - en "Java", utiliser les interfaces.
- 2 **Choisir ensuite le type (la classe) des attributs**
 - le plus adapté au problème à résoudre ;
 - exploiter les services offerts par les classes des bibliothèques de l'outil de développement ;
 - si remise en cause des choix (des types ou de l'algorithmie), respecter toujours le contrat (possible par l'utilisation des interfaces).
- 3 **Si besoin de faire évoluer le cahier des charges**
 - modifier le contrat en ajoutant des interfaces ;
 - implémenter ces nouvelles interfaces.

Les mécanismes de réutilisation

Principes d'une bonne conception "objet"

① Définir en premier les services que doit réaliser l'objet (fonctionnalités souhaitées)

- les définir avec précision ;
- programmation par "contrat" ;
- en "Java", utiliser les interfaces.

② Choisir ensuite le type (la classe) des attributs

- le plus adapté au problème à résoudre ;
- exploiter les services offerts par les classes des bibliothèques de l'outil de développement ;
- si remise en cause des choix (des types ou de l'algorithmie), respecter toujours le contrat (possible par l'utilisation des interfaces).

③ Si besoin de faire évoluer le cahier des charges

- modifier le contrat en ajoutant des interfaces ;
- implémenter ces nouvelles interfaces.

Les mécanismes de réutilisation

Principes d'une bonne conception "objet"

① Définir en premier les services que doit réaliser l'objet (fonctionnalités souhaitées)

- les définir avec précision ;
- programmation par "contrat" ;
- en "Java", utiliser les interfaces.

② Choisir ensuite le type (la classe) des attributs

- le plus adapté au problème à résoudre ;
- exploiter les services offerts par les classes des bibliothèques de l'outil de développement ;
- si remise en cause des choix (des types ou de l'algorithmie), respecter toujours le contrat (possible par l'utilisation des interfaces).

③ Si besoin de faire évoluer le cahier des charges

- modifier le contrat en ajoutant des interfaces ;
- implémenter ces nouvelles interfaces.

Les mécanismes de réutilisation

Principes d'une bonne conception "objet"

- 1 **Définir en premier les services que doit réaliser l'objet (fonctionnalités souhaitées)**
 - les définir avec précision ;
 - programmation par "contrat" ;
 - en "Java", utiliser les interfaces.
- 2 **Choisir ensuite le type (la classe) des attributs**
 - le plus adapté au problème à résoudre ;
 - exploiter les services offerts par les classes des bibliothèques de l'outil de développement ;
 - si remise en cause des choix (des types ou de l'algorithmie), respecter toujours le contrat (possible par l'utilisation des interfaces).
- 3 **Si besoin de faire évoluer le cahier des charges**
 - modifier le contrat en ajoutant des interfaces ;
 - implémenter ces nouvelles interfaces.

Les mécanismes de réutilisation

Principes d'une bonne conception "objet"

- 1 **Définir en premier les services que doit réaliser l'objet (fonctionnalités souhaitées)**
 - les définir avec précision ;
 - programmation par "contrat" ;
 - en "Java", utiliser les interfaces.
- 2 **Choisir ensuite le type (la classe) des attributs**
 - le plus adapté au problème à résoudre ;
 - exploiter les services offerts par les classes des bibliothèques de l'outil de développement ;
 - si remise en cause des choix (des types ou de l'algorithmie), respecter toujours le contrat (possible par l'utilisation des interfaces).
- 3 **Si besoin de faire évoluer le cahier des charges**
 - modifier le contrat en ajoutant des interfaces ;
 - implémenter ces nouvelles interfaces.

Les mécanismes de réutilisation

Principes d'une bonne conception "objet"

- 1 **Définir en premier les services que doit réaliser l'objet (fonctionnalités souhaitées)**
 - les définir avec précision ;
 - programmation par "contrat" ;
 - en "Java", utiliser les interfaces.
- 2 **Choisir ensuite le type (la classe) des attributs**
 - le plus adapté au problème à résoudre ;
 - exploiter les services offerts par les classes des bibliothèques de l'outil de développement ;
 - si remise en cause des choix (des types ou de l'algorithmie), respecter toujours le contrat (possible par l'utilisation des interfaces).
- 3 **Si besoin de faire évoluer le cahier des charges**
 - modifier le contrat en ajoutant des interfaces ;
 - implémenter ces nouvelles interfaces.

Les mécanismes de réutilisation

Principes d'une bonne conception "objet"

- 1 **Définir en premier les services que doit réaliser l'objet (fonctionnalités souhaitées)**
 - les définir avec précision ;
 - programmation par "contrat" ;
 - en "Java", utiliser les interfaces.
- 2 **Choisir ensuite le type (la classe) des attributs**
 - le plus adapté au problème à résoudre ;
 - exploiter les services offerts par les classes des bibliothèques de l'outil de développement ;
 - si remise en cause des choix (des types ou de l'algorithmie), respecter toujours le contrat (possible par l'utilisation des interfaces).
- 3 **Si besoin de faire évoluer le cahier des charges**
 - modifier le contrat en ajoutant des interfaces ;
 - implémenter ces nouvelles interfaces.

Les mécanismes de réutilisation

Principes d'une bonne conception "objet"

- 1 **Définir en premier les services que doit réaliser l'objet (fonctionnalités souhaitées)**
 - les définir avec précision ;
 - programmation par "contrat" ;
 - en "Java", utiliser les interfaces.
- 2 **Choisir ensuite le type (la classe) des attributs**
 - le plus adapté au problème à résoudre ;
 - exploiter les services offerts par les classes des bibliothèques de l'outil de développement ;
 - si remise en cause des choix (des types ou de l'algorithmie), respecter toujours le contrat (possible par l'utilisation des interfaces).
- 3 **Si besoin de faire évoluer le cahier des charges**
 - modifier le contrat en ajoutant des interfaces ;
 - implémenter ces nouvelles interfaces.

- 1 **Définir en premier les services que doit réaliser l'objet (fonctionnalités souhaitées)**
 - les définir avec précision ;
 - programmation par "contrat" ;
 - en "Java", utiliser les interfaces.
- 2 **Choisir ensuite le type (la classe) des attributs**
 - le plus adapté au problème à résoudre ;
 - exploiter les services offerts par les classes des bibliothèques de l'outil de développement ;
 - si remise en cause des choix (des types ou de l'algorithmie), respecter toujours le contrat (possible par l'utilisation des interfaces).
- 3 **Si besoin de faire évoluer le cahier des charges**
 - modifier le contrat en ajoutant des interfaces ;
 - implémenter ces nouvelles interfaces.

- ➊ **Définir en premier les services que doit réaliser l'objet (fonctionnalités souhaitées)**
 - les définir avec précision ;
 - programmation par "contrat" ;
 - en "Java", utiliser les interfaces.
- ➋ **Choisir ensuite le type (la classe) des attributs**
 - le plus adapté au problème à résoudre ;
 - exploiter les services offerts par les classes des bibliothèques de l'outil de développement ;
 - si remise en cause des choix (des types ou de l'algorithmie), respecter toujours le contrat (possible par l'utilisation des interfaces).
- ➌ **Si besoin de faire évoluer le cahier des charges**
 - modifier le contrat en ajoutant des interfaces ;
 - implémenter ces nouvelles interfaces.

Les mécanismes de réutilisation

Interfaces : intérêt

☞ Définir le contrat avec les "clients" d'une classe

- les services à assurer (notés ISvcXxxx dans nos projets).

☞ Retarder le choix du type associé aux attributs (abstraction)

- modifier l'implémentation d'une classe sans casser l'application car, avec l'expérience, on acquiert :
 - une meilleure connaissance des possibilités de langage
 - une utilisation des interfaces (traits, classes d'abstraction, ...)

☞ Mettre en œuvre le polymorphisme pour :

- disposer de plusieurs facettes de la même entité
PointGPS
- spécialiser des comportements

☞ Assurer le lien avec :

- des objets qui n'appartiennent pas à la même hiérarchie d'héritage
- le "Runtime - JVM" Java de la machine cible (événements, ...)

Les mécanismes de réutilisation

Interfaces : intérêt

☞ Définir le contrat avec les "clients" d'une classe

- les services à assurer (notés ISvcXxxx dans nos projets).

☞ Retarder le choix du type associé aux attributs (abstraction)

- modifier l'implémentation d'une classe sans casser l'application car, avec l'expérience, on acquiert :

• la maîtrise complète des possibilités de langage

• l'abstraction des comportements (traits, classes d'abstraction, ...)

☞ Mettre en œuvre le polymorphisme pour :

- disposer de plusieurs facettes de la même entité
PointGPS
- spécialiser des comportements

☞ Assurer le lien avec :

- des objets qui n'appartiennent pas à la même hiérarchie d'héritage
- le "Runtime - JVM" Java de la machine cible (événements, ...)

Les mécanismes de réutilisation

Interfaces : intérêt

- ☞ **Définir le contrat avec les "clients" d'une classe**
 - les services à assurer (notés ISvcXxxx dans nos projets).
- ☞ **Retarder le choix du type associé aux attributs (abstraction)**
 - modifier l'implémentation d'une classe sans casser l'application car, avec l'expérience, on acquiert :
 - ① meilleure connaissance des possibilités du langage
 - ② optimisation des performances (taille, vitesse d'exécution, ...)
- ☞ **Mettre en œuvre le polymorphisme pour :**
 - disposer de plusieurs facettes de la même entité
PointGPS
 - spécialiser des comportements
- ☞ **Assurer le lien avec :**
 - des objets qui n'appartiennent pas à la même hiérarchie d'héritage
 - le "Runtime - JVM" Java de la machine cible (événements, ...)

Les mécanismes de réutilisation

Interfaces : intérêt

- ☞ **Définir le contrat avec les "clients" d'une classe**
 - les services à assurer (notés ISvcXxxx dans nos projets).
- ☞ **Retarder le choix du type associé aux attributs (abstraction)**
 - modifier l'implémentation d'une classe sans casser l'application car, avec l'expérience, on acquiert :
 - 1 meilleure connaissance des possibilités du langage
 - 2 optimisation des performances (taille, vitesse d'exécution, ...)
- ☞ **Mettre en œuvre le polymorphisme pour :**
 - disposer de plusieurs facettes de la même entité
PointGPS
 - spécialiser des comportements
- ☞ **Assurer le lien avec :**
 - des objets qui n'appartiennent pas à la même hiérarchie d'héritage
 - le "Runtime - JVM" Java de la machine cible (événements, ...)

- ☞ **Définir le contrat avec les "clients" d'une classe**
 - les services à assurer (notés ISvcXxxx dans nos projets).
- ☞ **Retarder le choix du type associé aux attributs (abstraction)**
 - modifier l'implémentation d'une classe sans casser l'application car, avec l'expérience, on acquiert :
 - 1 meilleure connaissance des possibilités du langage
 - 2 optimisation des performances (taille, vitesse d'exécution, ...)
- ☞ **Mettre en œuvre le polymorphisme pour :**
 - disposer de plusieurs facettes de la même entité
PointGPS
 - spécialiser des comportements
- ☞ **Assurer le lien avec :**
 - des objets qui n'appartiennent pas à la même hiérarchie d'héritage
 - le "Runtime - JVM" Java de la machine cible (événements, ...)

- ☞ **Définir le contrat avec les "clients" d'une classe**
 - les services à assurer (notés ISvcXxxx dans nos projets).
- ☞ **Retarder le choix du type associé aux attributs (abstraction)**
 - modifier l'implémentation d'une classe sans casser l'application car, avec l'expérience, on acquiert :
 - 1 meilleure connaissance des possibilités du langage
 - 2 optimisation des performances (taille, vitesse d'exécution, ...)
- ☞ **Mettre en œuvre le polymorphisme pour :**
 - disposer de plusieurs facettes de la même entité
PointGPS
 - spécialiser des comportements
- ☞ **Assurer le lien avec :**
 - des objets qui n'appartiennent pas à la même hiérarchie d'héritage
 - le "Runtime - JVM" Java de la machine cible (événements, ...)

- ☞ **Définir le contrat avec les "clients" d'une classe**
 - les services à assurer (notés ISvcXxxx dans nos projets).
- ☞ **Retarder le choix du type associé aux attributs (abstraction)**
 - modifier l'implémentation d'une classe sans casser l'application car, avec l'expérience, on acquiert :
 - 1 meilleure connaissance des possibilités du langage
 - 2 optimisation des performances (taille, vitesse d'exécution, ...)
- ☞ **Mettre en œuvre le polymorphisme pour :**
 - disposer de plusieurs facettes de la même entité
PointGPS
 - spécialiser des comportements
 - factorisation
- ☞ **Assurer le lien avec :**
 - des objets qui n'appartiennent pas à la même hiérarchie d'héritage
 - le "Runtime - JVM" Java de la machine cible (événements, ...)

- ☞ **Définir le contrat avec les "clients" d'une classe**
 - les services à assurer (notés ISvcXxxx dans nos projets).
- ☞ **Retarder le choix du type associé aux attributs (abstraction)**
 - modifier l'implémentation d'une classe sans casser l'application car, avec l'expérience, on acquiert :
 - 1 meilleure connaissance des possibilités du langage
 - 2 optimisation des performances (taille, vitesse d'exécution, ...)
- ☞ **Mettre en œuvre le polymorphisme pour :**
 - disposer de plusieurs facettes de la même entité
PointGPS
 - spécialiser des comportements
 - factorisation
- ☞ **Assurer le lien avec :**
 - des objets qui n'appartiennent pas à la même hiérarchie d'héritage
 - le "Runtime - JVM" Java de la machine cible (événements, ...)

- ☞ **Définir le contrat avec les "clients" d'une classe**
 - les services à assurer (notés ISvcXxxx dans nos projets).
- ☞ **Retarder le choix du type associé aux attributs (abstraction)**
 - modifier l'implémentation d'une classe sans casser l'application car, avec l'expérience, on acquiert :
 - 1 meilleure connaissance des possibilités du langage
 - 2 optimisation des performances (taille, vitesse d'exécution, ...)
- ☞ **Mettre en œuvre le polymorphisme pour :**
 - disposer de plusieurs facettes de la même entité
PointGPS
 - spécialiser des comportements
 - factorisation
- ☞ **Assurer le lien avec :**
 - des objets qui n'appartiennent pas à la même hiérarchie d'héritage
 - le "Runtime - JVM" Java de la machine cible (événements, ...)

- ☞ **Définir le contrat avec les "clients" d'une classe**
 - les services à assurer (notés ISvcXxxx dans nos projets).
- ☞ **Retarder le choix du type associé aux attributs (abstraction)**
 - modifier l'implémentation d'une classe sans casser l'application car, avec l'expérience, on acquiert :
 - 1 meilleure connaissance des possibilités du langage
 - 2 optimisation des performances (taille, vitesse d'exécution, ...)
- ☞ **Mettre en œuvre le polymorphisme pour :**
 - disposer de plusieurs facettes de la même entité
PointGPS
 - spécialiser des comportements
 - factorisation
- ☞ **Assurer le lien avec :**
 - des objets qui n'appartiennent pas à la même hiérarchie d'héritage
 - le "Runtime - JVM" Java de la machine cible (événements, ...)

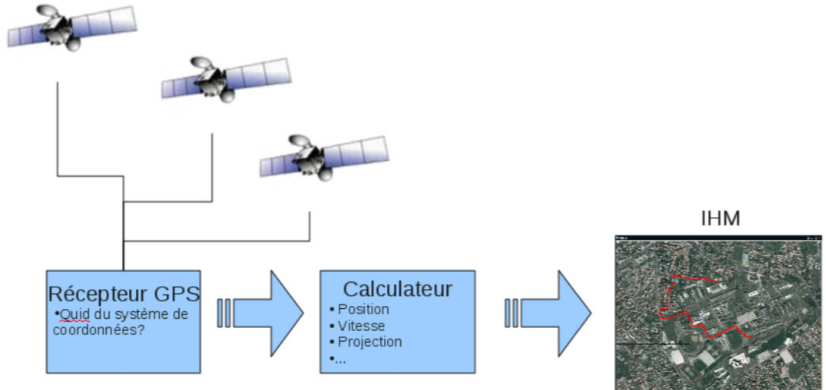
- ☞ **Définir le contrat avec les "clients" d'une classe**
 - les services à assurer (notés ISvcXxxx dans nos projets).
- ☞ **Retarder le choix du type associé aux attributs (abstraction)**
 - modifier l'implémentation d'une classe sans casser l'application car, avec l'expérience, on acquiert :
 - 1 meilleure connaissance des possibilités du langage
 - 2 optimisation des performances (taille, vitesse d'exécution, ...)
- ☞ **Mettre en œuvre le polymorphisme pour :**
 - disposer de plusieurs facettes de la même entité
PointGPS
 - spécialiser des comportements
 - factorisation
- ☞ **Assurer le lien avec :**
 - des objets qui n'appartiennent pas à la même hiérarchie d'héritage
 - le "Runtime - JVM" Java de la machine cible (événements, ...)

- ☞ **Définir le contrat avec les "clients" d'une classe**
 - les services à assurer (notés ISvcXxxx dans nos projets).
- ☞ **Retarder le choix du type associé aux attributs (abstraction)**
 - modifier l'implémentation d'une classe sans casser l'application car, avec l'expérience, on acquiert :
 - ① meilleure connaissance des possibilités du langage
 - ② optimisation des performances (taille, vitesse d'exécution, ...)
- ☞ **Mettre en œuvre le polymorphisme pour :**
 - disposer de plusieurs facettes de la même entité
PointGPS
 - spécialiser des comportements
 - factorisation
- ☞ **Assurer le lien avec :**
 - des objets qui n'appartiennent pas à la même hiérarchie d'héritage
 - le "Runtime - JVM" Java de la machine cible (événements, ...)

- ☞ **Définir le contrat avec les "clients" d'une classe**
 - les services à assurer (notés ISvcXxxx dans nos projets).
- ☞ **Retarder le choix du type associé aux attributs (abstraction)**
 - modifier l'implémentation d'une classe sans casser l'application car, avec l'expérience, on acquiert :
 - 1 meilleure connaissance des possibilités du langage
 - 2 optimisation des performances (taille, vitesse d'exécution, ...)
- ☞ **Mettre en œuvre le polymorphisme pour :**
 - disposer de plusieurs facettes de la même entité
PointGPS
 - spécialiser des comportements
 - factorisation
- ☞ **Assurer le lien avec :**
 - des objets qui n'appartiennent pas à la même hiérarchie d'héritage
 - le "Runtime - JVM" Java de la machine cible (événements, ...)

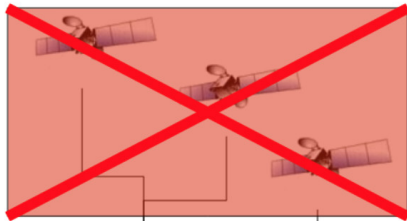
Les mécanismes de réutilisation

Etude de cas : Modélisation d'un GPS



Les mécanismes de réutilisation

Etude de cas : Modélisation d'un GPS



```
PRN 1 0.0 0.0 0.0 0.0 0.0 0.0
PRN 2 0.0 0.0 0.0 0.0 0.0 0.0
PRN 3 0.0 0.0 0.0 0.0 0.0 0.0
PRN 4 0.0 0.0 0.0 0.0 0.0 0.0
PRN 5 0.0 0.0 0.0 0.0 0.0 0.0
PRN 6 0.0 0.0 0.0 0.0 0.0 0.0
PRN 7 0.0 0.0 0.0 0.0 0.0 0.0
PRN 8 0.0 0.0 0.0 0.0 0.0 0.0
PRN 9 0.0 0.0 0.0 0.0 0.0 0.0
PRN 10 0.0 0.0 0.0 0.0 0.0 0.0
PRN 11 0.0 0.0 0.0 0.0 0.0 0.0
PRN 12 0.0 0.0 0.0 0.0 0.0 0.0
PRN 13 0.0 0.0 0.0 0.0 0.0 0.0
PRN 14 0.0 0.0 0.0 0.0 0.0 0.0
PRN 15 0.0 0.0 0.0 0.0 0.0 0.0
PRN 16 0.0 0.0 0.0 0.0 0.0 0.0
PRN 17 0.0 0.0 0.0 0.0 0.0 0.0
PRN 18 0.0 0.0 0.0 0.0 0.0 0.0
PRN 19 0.0 0.0 0.0 0.0 0.0 0.0
PRN 20 0.0 0.0 0.0 0.0 0.0 0.0
PRN 21 0.0 0.0 0.0 0.0 0.0 0.0
PRN 22 0.0 0.0 0.0 0.0 0.0 0.0
PRN 23 0.0 0.0 0.0 0.0 0.0 0.0
PRN 24 0.0 0.0 0.0 0.0 0.0 0.0
PRN 25 0.0 0.0 0.0 0.0 0.0 0.0
PRN 26 0.0 0.0 0.0 0.0 0.0 0.0
PRN 27 0.0 0.0 0.0 0.0 0.0 0.0
PRN 28 0.0 0.0 0.0 0.0 0.0 0.0
PRN 29 0.0 0.0 0.0 0.0 0.0 0.0
PRN 30 0.0 0.0 0.0 0.0 0.0 0.0
PRN 31 0.0 0.0 0.0 0.0 0.0 0.0
PRN 32 0.0 0.0 0.0 0.0 0.0 0.0
PRN 33 0.0 0.0 0.0 0.0 0.0 0.0
PRN 34 0.0 0.0 0.0 0.0 0.0 0.0
PRN 35 0.0 0.0 0.0 0.0 0.0 0.0
PRN 36 0.0 0.0 0.0 0.0 0.0 0.0
PRN 37 0.0 0.0 0.0 0.0 0.0 0.0
PRN 38 0.0 0.0 0.0 0.0 0.0 0.0
PRN 39 0.0 0.0 0.0 0.0 0.0 0.0
PRN 40 0.0 0.0 0.0 0.0 0.0 0.0
PRN 41 0.0 0.0 0.0 0.0 0.0 0.0
PRN 42 0.0 0.0 0.0 0.0 0.0 0.0
PRN 43 0.0 0.0 0.0 0.0 0.0 0.0
PRN 44 0.0 0.0 0.0 0.0 0.0 0.0
PRN 45 0.0 0.0 0.0 0.0 0.0 0.0
PRN 46 0.0 0.0 0.0 0.0 0.0 0.0
PRN 47 0.0 0.0 0.0 0.0 0.0 0.0
PRN 48 0.0 0.0 0.0 0.0 0.0 0.0
PRN 49 0.0 0.0 0.0 0.0 0.0 0.0
PRN 50 0.0 0.0 0.0 0.0 0.0 0.0
```

Fichiers de coordonnées
• Lambert 2
• WGS84

Récepteur GPS
• Quid du système de coordonnées?

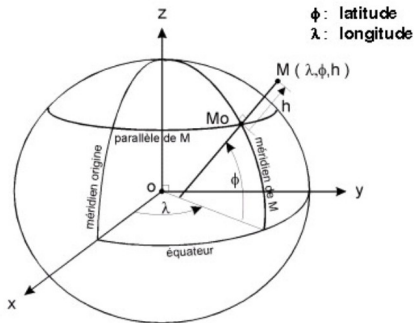


Calculateur
• Position
• Vitesse
• Projection
• ...



Les mécanismes de réutilisation

Etude de cas : Un point (GPS) à la base c'est quoi ?



3 représentations en WGS84 :

- ☞ latitude, longitude en degré décimal : double
- ☞ latitude, longitude en degré sexagésimal : entier
- ☞ latitude, longitude en radian : double

Le contrat :

- ☞ stocker une position GPS sous forme latitude, longitude en degré décimal
- ☞ connaître la position dans un format "**latitude, longitude en degré décimal**"
- ☞ stocker une position GPS sous forme latitude, longitude en degré sexagésimal
- ☞ connaître la position dans un format "**latitude, longitude en degré sexagésimal**"
- ☞ stocker une position GPS sous forme latitude, longitude en radian
- ☞ connaître la position dans un format "**latitude, longitude en radian**"

Le contrat :

- ☞ stocker une position GPS sous forme latitude, longitude en degré décimal
- ☞ connaître la position dans un format "**latitude, longitude en degré décimal**"
- ☞ stocker une position GPS sous forme latitude, longitude en degré sexagésimal
- ☞ connaître la position dans un format "**latitude, longitude en degré sexagésimal**"
- ☞ stocker une position GPS sous forme latitude, longitude en radian
- ☞ connaître la position dans un format "**latitude, longitude en radian**"

Le contrat :

- ☞ stocker une position GPS sous forme latitude, longitude en degré décimal
- ☞ connaître la position dans un format "**latitude, longitude en degré décimal**"
- ☞ stocker une position GPS sous forme latitude, longitude en degré sexagésimal
- ☞ connaître la position dans un format "**latitude, longitude en degré sexagésimal**"
- ☞ stocker une position GPS sous forme latitude, longitude en radian
- ☞ connaître la position dans un format "**latitude, longitude en radian**"

Le contrat :

- ☞ stocker une position GPS sous forme latitude, longitude en degré décimal
- ☞ connaître la position dans un format "**latitude, longitude en degré décimal**"
- ☞ stocker une position GPS sous forme latitude, longitude en degré sexagésimal
- ☞ connaître la position dans un format "**latitude, longitude en degré sexagésimal**"
- ☞ stocker une position GPS sous forme latitude, longitude en radian
- ☞ connaître la position dans un format "**latitude, longitude en radian**"

Le contrat :

- ☞ stocker une position GPS sous forme latitude, longitude en degré décimal
- ☞ connaître la position dans un format "**latitude, longitude en degré décimal**"
- ☞ stocker une position GPS sous forme latitude, longitude en degré sexagésimal
- ☞ connaître la position dans un format "**latitude, longitude en degré sexagésimal**"
- ☞ stocker une position GPS sous forme latitude, longitude en radian
- ☞ connaître la position dans un format "**latitude, longitude en radian**"

Le contrat :

- ☞ stocker une position GPS sous forme latitude, longitude en degré décimal
- ☞ connaître la position dans un format "**latitude, longitude en degré décimal**"
- ☞ stocker une position GPS sous forme latitude, longitude en degré sexagésimal
- ☞ connaître la position dans un format "**latitude, longitude en degré sexagésimal**"
- ☞ stocker une position GPS sous forme latitude, longitude en radian
- ☞ connaître la position dans un format "**latitude, longitude en radian**"

Les mécanismes de réutilisation

Etude de cas : Notation UML de l'interface ISvcPointGps

« ISvcPointGps »

- + *setLatitudeDegDec(latitude : double) : void*
- + *getLatitudeDegDec() : double*
- + *setLatitudeDegSexa(degre : int, min : int, sec : int) : void*
- + *getLatitudeDegSexa() : int[]*
- + *setLatitudeRad(latitude : double) : void*
- + *getLatitudeRad() : double*
- + *setLongitudeDegDec(longitude : double) : void*
- + *getLongitudeDegDec() : double*
- + *setLongitudeDegSexa(degre : int, min : int, sec : int) : void*
- + *getLongitudeDegSexa() : int[]*
- + *setLongitudeRad(longitude : double) : void*
- + *getLongitudeRad() : double*

rmq : abstrait (abstract, noté en italique dans UML)

Les mécanismes de réutilisation

Codage de l'interface : ISvcPointGPS

```
public interface ISvcPointGPS
{
    public abstract void setLatitudeDegDec(double latitude);
    public abstract double getLatitudeDegDec();
    public abstract void setLongitudeDegDec(double longitude);
    public abstract double getLongitudeDegDec();
    public abstract void setLatitudeRad(double latitude);
    public abstract double getLatitudeRad();
    public abstract void setLongitudeRad(double longitude);
    public abstract double getLongitudeRad();
    public abstract void setLatitudeDegSexa(int degre, int min, int sec);
    public abstract int[] getLatitudeDegSexa();
    public abstract void setLongitudeDegSexa(int degre, int min, int sec);
    public abstract int[] getLongitudeDegSexa();
}
```

Exercice 2

Enoncé

1. Créer un nouveau projet BlueJ et saisir l'interface *ISvcPointGPS*.
2. Créer 3 classes *PointDegDec*, *PointDegSexa*, et *PointDegRad* qui implémentent (toutes les méthodes) de l'interface *ISvcPointGPS*.
3. Effectuer les tests pour vous assurer du codage correct des classes.