Programmation objet en Java

Vincent Vidal

Maître de Conférences
Enseignements : IUT Lyon 1 - pôle AP - Licence ESSIR - bureau 2ème étage
Recherche : Laboratoire LIRIS - bât. Nautibus
E-mail : vincent.vidal@univ-lyon1.fr

Supports de cours et TPs : `http://spiralconnect.univ-lyon1.fr` module "S2 - M2103 - Java et POO"
48H prévues $\approx$ 39H de cours-TDs + 6H de TPs, 1H - interros, et 2H - examen final

Évaluation : Contrôle continu + examen final + Bonus/Malus TP

Support de cours et livre de référence

- Supports de cours disponibles sur spiralconnect (`http://spiralconnect.univ-lyon1.fr`).
- Livre *Programmer en Java* de Claude Delannoy.

Plan

Historique du Java

Java est un terme argotique signifiant **café** en Amérique du Nord.

Java est

- un **langage de programmation** orienté objet.
- un **environnement d'exécution** (une machine virtuelle JVM) qui garantit la portabilité de Java.

Historique du Java

Java est un terme argotique signifiant **café** en Amérique du Nord.

Java est

- un **langage de programmation** orienté objet.
- un **environnement d'exécution** (une machine virtuelle JVM) qui garantit la portabilité de Java.

Historique du Java

- **Naissance en 1991** chez SUN Microsystems pour du code embarqué (James Gosling et Patrick Naughton).
- Syntaxe **proche du C++** (donc du C).
- Traduire ce langage en un langage universel (pas du code machine !), le **bytecode**.
- **Machine virtuelle** qui interprète le bytecode (et compile certaines parties).

Historique du Java

- **Naissance en 1991** chez SUN Microsystems pour du code embarqué (James Gosling et Patrick Naughton).
- Syntaxe **proche du C++** (donc du C).
- Traduire ce langage en un langage universel (pas du code machine !), le **bytecode**.
- **Machine virtuelle** qui interprète le bytecode (et compile certaines parties).

Historique du Java

- **Naissance en 1991** chez SUN Microsystems pour du code embarqué (James Gosling et Patrick Naughton).
- Syntaxe **proche du C++** (donc du C).
- Traduire ce langage en un langage universel (pas du code machine !), le **bytecode**.
- **Machine virtuelle** qui interprète le bytecode (et compile certaines parties).

Historique du Java

- **Naissance en 1991** chez SUN Microsystems pour du code embarqué (James Gosling et Patrick Naughton).
- Syntaxe **proche du C++** (donc du C).
- Traduire ce langage en un langage universel (pas du code machine !), le **bytecode**.
- **Machine virtuelle** qui interprète le bytecode (et compile certaines parties).

Java en quelques chiffres

- 97% des machines d'entreprises ont une JVM installée.
- Java est téléchargé plus d'un milliards de fois chaque année.
- Il y a plus de 9 millions de développeurs Java dans le monde.
- Tous les lecteurs de Blu-Ray utilisent Java.
- Plus de 1,4 milliards de cartes à puce utilisant Java sont produites chaque année.
- ...

Java en quelques chiffres

- 97% des machines d'entreprises ont une JVM installée.
- Java est téléchargé plus d'un milliards de fois chaque année.
- Il y a plus de 9 millions de développeurs Java dans le monde.
- Tous les lecteurs de Blu-Ray utilisent Java.
- Plus de 1,4 milliards de cartes à puce utilisant Java sont produites chaque année.
- ...

Java en quelques chiffres

- 97% des machines d'entreprises ont une JVM installée.
- Java est téléchargé plus d'un milliards de fois chaque année.
- Il y a plus de 9 millions de développeurs Java dans le monde.
- Tous les lecteurs de Blu-Ray utilisent Java.
- Plus de 1,4 milliards de cartes à puce utilisant Java sont produites chaque année.
- ...

Java en quelques chiffres

- 97% des machines d'entreprises ont une JVM installée.
- Java est téléchargé plus d'un milliards de fois chaque année.
- Il y a plus de 9 millions de développeurs Java dans le monde.
- Tous les lecteurs de Blu-Ray utilisent Java.
- Plus de 1,4 milliards de cartes à puce utilisant Java sont produites chaque année.
- ...

Java en quelques chiffres

- 97% des machines d'entreprises ont une JVM installée.
- Java est téléchargé plus d'un milliards de fois chaque année.
- Il y a plus de 9 millions de développeurs Java dans le monde.
- Tous les lecteurs de Blu-Ray utilisent Java.
- Plus de 1,4 milliards de cartes à puce utilisant Java sont produites chaque année.
- ...

Java en quelques chiffres

- 97% des machines d'entreprises ont une JVM installée.
- Java est téléchargé plus d'un milliards de fois chaque année.
- Il y a plus de 9 millions de développeurs Java dans le monde.
- Tous les lecteurs de Blu-Ray utilisent Java.
- Plus de 1,4 milliards de cartes à puce utilisant Java sont produites chaque année.
- ...

Ce que Java permet de faire

- Développer **des applications** (de bureau et d'entreprise).
- Développer **des sites Web** (intégration dans une page HTML)
  - Développer des applets (Java) pour vos sites Web (côté client).
  - Développer des sites en JSP (JavaServerPages, côté serveur).
- Développer **des applications** pour téléphones mobiles/tablettes **Androïd** et des applications pour des appareils embarqués.

Ce que Java permet de faire

- Développer **des applications** (de bureau et d'entreprise).
- Développer **des sites Web** (intégration dans une page HTML)
  - Développer des applets (Java) pour vos sites Web (côté client).
  - Développer des sites en JSP (JavaServerPages, côté serveur).
- Développer **des applications** pour téléphones mobiles/tablettes **Androïd** et des applications pour des appareils embarqués.

Ce que Java permet de faire

- Développer **des applications** (de bureau et d'entreprise).
- Développer **des sites Web** (intégration dans une page HTML)
  - Développer des applets (Java) pour vos sites Web (côté client).
  - Développer des sites en JSP (JavaServerPages, côté serveur).
- Développer **des applications** pour téléphones mobiles/tablettes **Androïd** et des applications pour des appareils embarqués.

Ce que Java permet de faire

- Développer **des applications** (de bureau et d'entreprise).
- Développer **des sites Web** (intégration dans une page HTML)
  - Développer des applets (Java) pour vos sites Web (côté client).
  - Développer des sites en JSP (JavaServerPages, côté serveur).
- Développer **des applications** pour téléphones mobiles/tablettes **Androïd** et des applications pour des appareils embarqués.

Ce que Java permet de faire

- Développer **des applications** (de bureau et d'entreprise).
- Développer **des sites Web** (intégration dans une page HTML)
  - Développer des applets (Java) pour vos sites Web (côté client).
  - Développer des sites en JSP (JavaServerPages, côté serveur).
- Développer **des applications** pour téléphones mobiles/tablettes **Androïd** et des applications pour des appareils embarqués.

Les clefs du succès

- Java est orienté objet, mais **plus simple que le C++**.
- La gestion des interfaces graphiques est intégrée dans Java (Swing, JavaFX en plus pour Java 8).
- Des applications et du Web (applets, JSP).
- La **portabilité** : le même bytecode s'exécute de la même façon avec la même précision quel que soit l'environnement.
- Java assure la gestion mémoire : **aucune fuite mémoire possible**.
- **Java est sûr** : il ne peut pas y avoir d'accès direct à la mémoire non-autorisé + détections des variables non-initialisées et utilisées. Le temps de débogage est relativement court.
- Java utilise l'encodage unicode des caractères sur 2 octets : é, è, à...

Les clefs du succès

- Java est orienté objet, mais **plus simple que le C++**.
- La gestion des interfaces graphiques est intégrée dans Java (Swing, JavaFX en plus pour Java 8).
- Des applications et du Web (applets, JSP).
- La **portabilité** : le même bytecode s'exécute de la même façon avec la même précision quel que soit l'environnement.
- Java assure la gestion mémoire : **aucune fuite mémoire possible**.
- **Java est sûr** : il ne peut pas y avoir d'accès direct à la mémoire non-autorisé + détections des variables non-initialisées et utilisées. Le temps de débogage est relativement court.
- Java utilise l'encodage unicode des caractères sur 2 octets : é, è, à...

Les clefs du succès

- Java est orienté objet, mais **plus simple que le C++**.
- La gestion des interfaces graphiques est intégrée dans Java (Swing, JavaFX en plus pour Java 8).
- Des applications et du Web (applets, JSP).
- La **portabilité** : le même bytecode s'exécute de la même façon avec la même précision quel que soit l'environnement.
- Java assure la gestion mémoire : **aucune fuite mémoire possible**.
- **Java est sûr** : il ne peut pas y avoir d'accès direct à la mémoire non-autorisé + détections des variables non-initialisées et utilisées. Le temps de débogage est relativement court.
- Java utilise l'encodage unicode des caractères sur 2 octets : é, è, à...

Les clefs du succès

- Java est orienté objet, mais **plus simple que le C++**.
- La gestion des interfaces graphiques est intégrée dans Java (Swing, JavaFX en plus pour Java 8).
- Des applications et du Web (applets, JSP).
- La **portabilité** : le même bytecode s'exécute de la même façon avec la même précision quel que soit l'environnement.
- Java assure la gestion mémoire : **aucune fuite mémoire possible**.
- **Java est sûr** : il ne peut pas y avoir d'accès direct à la mémoire non-autorisé + détections des variables non-initialisées et utilisées. Le temps de débogage est relativement court.
- Java utilise l'encodage unicode des caractères sur 2 octets : é, è, à...

Les clefs du succès

- Java est orienté objet, mais **plus simple que le C++**.
- La gestion des interfaces graphiques est intégrée dans Java (Swing, JavaFX en plus pour Java 8).
- Des applications et du Web (applets, JSP).
- La **portabilité** : le même bytecode s'exécute de la même façon avec la même précision quel que soit l'environnement.
- Java assure la gestion mémoire : **aucune fuite mémoire possible**.
- **Java est sûr** : il ne peut pas y avoir d'accès direct à la mémoire non-autorisé + détections des variables non-initialisées et utilisées. Le temps de débogage est relativement court.
- Java utilise l'encodage unicode des caractères sur 2 octets : é, è, à...

Les clefs du succès

- Java est orienté objet, mais **plus simple que le C++**.
- La gestion des interfaces graphiques est intégrée dans Java (Swing, JavaFX en plus pour Java 8).
- Des applications et du Web (applets, JSP).
- La **portabilité** : le même bytecode s'exécute de la même façon avec la même précision quel que soit l'environnement.
- Java assure la gestion mémoire : **aucune fuite mémoire possible**.
- **Java est sûr** : il ne peut pas y avoir d'accès direct à la mémoire non-autorisé + détections des variables non-initialisées et utilisées. Le temps de débogage est relativement court.
- Java utilise l'encodage unicode des caractères sur 2 octets : é, è, à...

Les clefs du succès

- Java est orienté objet, mais **plus simple que le C++**.
- La gestion des interfaces graphiques est intégrée dans Java (Swing, JavaFX en plus pour Java 8).
- Des applications et du Web (applets, JSP).
- La **portabilité** : le même bytecode s'exécute de la même façon avec la même précision quel que soit l'environnement.
- Java assure la gestion mémoire : **aucune fuite mémoire possible**.
- **Java est sûr** : il ne peut pas y avoir d'accès direct à la mémoire non-autorisé + détections des variables non-initialisées et utilisées. Le temps de débogage est relativement court.
- Java utilise l'encodage unicode des caractères sur 2 octets : é, è, à...

Qu'est-ce qu'un objet ?

Définition

Un objet est une représentation abstraite et autonome qui :

- a un état particulier à un instant t (*il "contient" des données*) ;
- a un/des comportements possibles à partir de cet état (*il "contient" des fonctionnalités*).

Remarque : En programmation fonctionnelle (C), les données et les traitements sur les données sont séparés, ce n'est plus le cas en programmation objet.

Qu'est-ce qu'un objet ?

Définition

Un objet est une représentation abstraite et autonome qui :

- a un état particulier à un instant t (*il "contient" des données*) ;
- a un/des comportements possibles à partir de cet état (*il "contient" des fonctionnalités*).

Remarque : En programmation fonctionnelle (C), les données et les traitements sur les données sont séparés, ce n'est plus le cas en programmation objet.

Qu'est-ce qu'un objet ?

Définition

Un objet est une représentation abstraite et autonome qui :

- a un état particulier à un instant t (*il "contient" des données*) ;
- a un/des comportements possibles à partir de cet état (*il "contient" des fonctionnalités*).

Remarque : En programmation fonctionnelle (C), les données et les traitements sur les données sont séparés, ce n'est plus le cas en programmation objet.

Qu'est-ce qu'un objet ?

Un objet peut être vu comme la généralisation de la notion de variable à des types abstraits ayant souvent un lien fort avec un type concret (du monde réel ou lié à notre problème).

Exemples :

Chien c : c est un objet de type Chien muni des données internes "race, age" et muni de la fonctionnalité "wouaf()"

Table t : t est un objet de type Table muni des données internes "typeBois, couleur, prix" et muni de la fonctionnalité "changerCouleur(nouvelleCouleur)"

Qu'est-ce qu'un objet ?

Un objet peut être vu comme la généralisation de la notion de variable à des types abstraits ayant souvent un lien fort avec un type concret (du monde réel ou lié à notre problème).

Exemples :
Chien c : c est un objet de type Chien muni des données internes "race, age" et muni de la fonctionnalité "wouaf()"

Table t : t est un objet de type Table muni des données internes "typeBois, couleur, prix" et muni de la fonctionnalité "changerCouleur(nouvelleCouleur)"

Qu'est-ce qu'un objet ?

Un objet peut être vu comme la généralisation de la notion de variable à des types abstraits ayant souvent un lien fort avec un type concret (du monde réel ou lié à notre problème).

Exemples :
Chien c : c est un objet de type Chien muni des données internes "race, age" et muni de la fonctionnalité "wouaf()"

Table t : t est un objet de type Table muni des données internes "typeBois, couleur, prix" et muni de la fonctionnalité "changer-Couleur(nouvelleCouleur)"

Qu'est-ce qu'une classe ?

Définition

Une classe est un modèle pour représenter *les données* et *les services* d'un objet qui :

- permet de spécifier les caractéristiques/propriétés de l'objet ;
- permet de contruire des objets de son type afin qu'ils soient dans un état initial valide.

Une classe est le type d'un ou de plusieurs objet. C'est cette classe qui définit les caractéristiques des objets de son type.

Qu'est-ce qu'une classe ?

Définition

Une classe est un modèle pour représenter *les données* et *les services* d'un objet qui :

- permet de spécifier les caractéristiques/propriétés de l'objet ;
- permet de contruire des objets de son type afin qu'ils soient dans un état initial valide.

Une classe est le type d'un ou de plusieurs objet. C'est cette classe qui définit les caractéristiques des objets de son type.

Qu'est-ce qu'une classe ?

Définition

Une classe est un modèle pour représenter *les données* et *les services* d'un objet qui :

- permet de spécifier les caractéristiques/propriétés de l'objet ;
- permet de contruire des objets de son type afin qu'ils soient dans un état initial valide.

Une classe est le type d'un ou de plusieurs objet. C'est cette classe qui définit les caractéristiques des objets de son type.

Exemple

Un objet c de type Chrono : c.start(), c.stop(), c.affiche()... ;

C'est l'écriture de la classe Chrono qui va déterminer les fonctionnalités (start() etc.) ainsi que la représentation interne du temps pour les objets de type Chrono.

Exemple

Un objet c de type Chrono : c.start(), c.stop(), c.affiche()... ;

C'est l'écriture de la classe Chrono qui va déterminer les fonctionnalités (start() etc.) ainsi que la représentation interne du temps pour les objets de type Chrono.

Pourquoi la POO ?

- Elle se fonde sur la programmation structurée/procédurale (e.g. le C).
- Elle apporte des avantages conséquents :
  - plus de **fiabilité** et de **maintenabilité** dans les logiciels, notamment grâce à l'**encapsulation des données** au sein d'un ***objet*** et à l'automatisation de l'initialisation avec les **constructeurs** ;
  - **réutilisation** du code existant facilitée, notamment grâce à l'**héritage** et l'**encapsulation des données**.

Java est un langage pur objet : un programme sera formé d'une ***classe*** ou de la réunion de plusieurs classes et il instanciera des objets. *Cette pureté est troublée par l'existence de types primitifs non-objets* (entiers, flottants, caractères et booléens).

Pourquoi la POO ?

- Elle se fonde sur la programmation structurée/procédurale (e.g. le C).
- Elle apporte des avantages conséquents :
  - plus de **fiabilité** et de **maintenabilité** dans les logiciels, notamment grâce à l'**encapsulation des données** au sein d'un ***objet*** et à l'automatisation de l'initialisation avec les **constructeurs** ;
  - **réutilisation** du code existant facilitée, notamment grâce à l'**héritage** et l'**encapsulation des données**.

Java est un langage pur objet : un programme sera formé d'une ***classe*** ou de la réunion de plusieurs classes et il instanciera des objets. *Cette pureté est troublée par l'existence de types primitifs non-objets* (entiers, flottants, caractères et booléens).

Pourquoi la POO ?

- Elle se fonde sur la programmation structurée/procédurale (e.g. le C).
- Elle apporte des avantages conséquents :
  - plus de **fiabilité** et de **maintenabilité** dans les logiciels, notamment grâce à l'**encapsulation des données** au sein d'un ***objet*** et à l'automatisation de l'initialisation avec les **constructeurs** ;
  - **réutilisation** du code existant facilitée, notamment grâce à l'**héritage** et l'**encapsulation des données**.

Java est un langage pur objet : un programme sera formé d'une ***classe*** ou de la réunion de plusieurs classes et il instanciera des objets. *Cette pureté est troublée par l'existence de types primitifs non-objets* (entiers, flottants, caractères et booléens).

Pourquoi la POO ?

- Elle se fonde sur la programmation structurée/procédurale (e.g. le C).
- Elle apporte des avantages conséquents :
  - plus de **fiabilité** et de **maintenabilité** dans les logiciels, notamment grâce à l'**encapsulation des données** au sein d'un ***objet*** et à l'automatisation de l'initialisation avec les **constructeurs** ;
  - **réutilisation** du code existant facilitée, notamment grâce à l'**héritage** et l'**encapsulation des données**.

Java est un langage pur objet : un programme sera formé d'une ***classe*** ou de la réunion de plusieurs classes et il instanciera des objets. *Cette pureté est troublée par l'existence de types primitifs non-objets* (entiers, flottants, caractères et booléens).

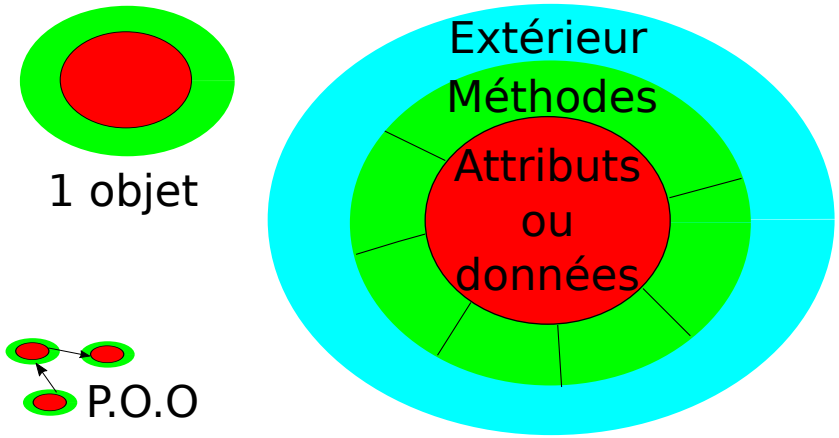
Pourquoi la POO ?

- Elle se fonde sur la programmation structurée/procédurale (e.g. le C).
- Elle apporte des avantages conséquents :
  - plus de **fiabilité** et de **maintenabilité** dans les logiciels, notamment grâce à l'**encapsulation des données** au sein d'un ***objet*** et à l'automatisation de l'initialisation avec les **constructeurs** ;
  - **réutilisation** du code existant facilitée, notamment grâce à l'**héritage** et l'**encapsulation des données**.

Java est un langage pur objet : un programme sera formé d'une ***classe*** ou de la réunion de plusieurs classes et il instanciera des objets. *Cette pureté est troublée par l'existence de types primitifs non-objets* (entiers, flottants, caractères et booléens).

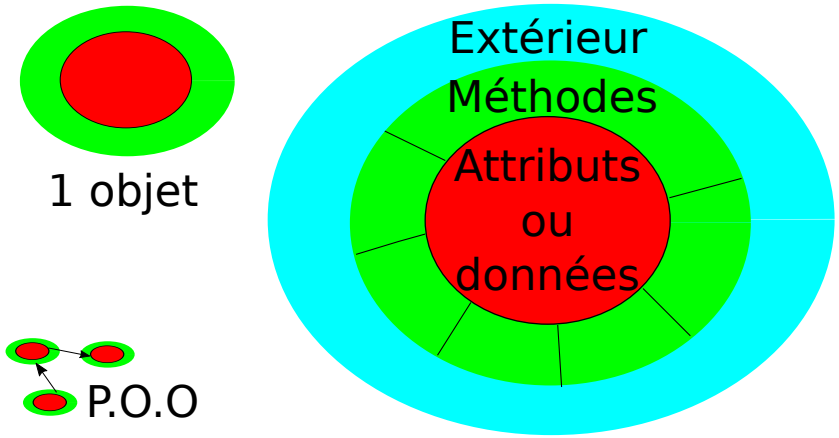
Pourquoi l'encapsulation ?

- L'**encapsulation des données** signifie qu'il n'est pas possible d'agir directement sur les données d'un objet, il faut passer forcément pas les *méthodes* (≈ fonctions) accessibles de l'objets ; cela diminue les sources de bogue.



Pourquoi l'encapsulation ?

- L'**encapsulation des données** signifie qu'il n'est pas possible d'agir directement sur les données d'un objet, il faut passer forcément pas les *méthodes* (≈ fonctions) accessibles de l'objets ; cela diminue les sources de bogue.



Pourquoi l'encapsulation ?

- Vue de l'extérieur, **on ne connaît pas la représentation des données au sein d'un objet**, on connaît seulement l'ensemble des *méthodes* ($\approx$ fonctions) disponibles sur cet objet : on réalise ce qu'on appelle une *abstraction des données*.

Ainsi une modification de la représentation des données n'a pas de répercussion sur les utilisateurs de l'objet. **Tout ce qui est caché peut être modifié sans gêner les utilisateurs.**

Pourquoi l'encapsulation ?

- Vue de l'extérieur, **on ne connaît pas la représentation des données au sein d'un objet**, on connaît seulement l'ensemble des *méthodes* ($\approx$ fonctions) disponibles sur cet objet : on réalise ce qu'on appelle une *abstraction des données*.

  Ainsi une modification de la représentation des données n'a pas de répercussion sur les utilisateurs de l'objet. **Tout ce qui est caché peut être modifié sans gêner les utilisateurs**.

Les caractéristiques d'une approche POO pure pour la résolution d'un problème

- **Les objets sont des variables améliorées** qui reçoivent des requêtes/messages et se débrouillent pour les traiter ; *un message est un appel de fonction* ;
- Un programme est un ensemble d'objets qui s'envoient des requêtes/messages.
- **Chaque objet à son propre espace mémoire** (il en est le propriétaire !), un objet pouvant contenir d'autres objets ;
- **Chaque objet est d'un type précis** et c'est son type (sa classe) qui caractérise les messages qu'il peut traiter ;
- **Des objets de types différents peuvent être d'un même type plus général** (*regroupant les caractéristiques et comportements en commun*) et comprendre les messages adressés pour le type général.

Les caractéristiques d'une approche POO pure pour la résolution d'un problème

- **Les objets sont des variables améliorées** qui reçoivent des requêtes/messages et se débrouillent pour les traiter ; *un message est un appel de fonction* ;
- Un programme est un ensemble d'objets qui s'envoient des requêtes/messages.
- **Chaque objet à son propre espace mémoire** (il en est le propriétaire !), un objet pouvant contenir d'autres objets ;
- **Chaque objet est d'un type précis** et c'est son type (sa classe) qui caractérise les messages qu'il peut traiter ;
- **Des objets de types différents peuvent être d'un même type plus général** (*regroupant les caractéristiques et comportements en commun*) et comprendre les messages adressés pour le type général.

Les caractéristiques d'une approche POO pure pour la résolution d'un problème

- **Les objets sont des variables améliorées** qui reçoivent des requêtes/messages et se débrouillent pour les traiter ; *un message est un appel de fonction* ;
- Un programme est un ensemble d'objets qui s'envoient des requêtes/messages.
- **Chaque objet à son propre espace mémoire** (il en est le propriétaire !), un objet pouvant contenir d'autres objets ;
- **Chaque objet est d'un type précis** et c'est son type (sa classe) qui caractérise les messages qu'il peut traiter ;
- **Des objets de types différents peuvent être d'un même type plus général** (*regroupant les caractéristiques et comportements en commun*) et comprendre les messages adressés pour le type général.

Les caractéristiques d'une approche POO pure pour la résolution d'un problème

- **Les objets sont des variables améliorées** qui reçoivent des requêtes/messages et se débrouillent pour les traiter ; *un message est un appel de fonction* ;
- Un programme est un ensemble d'objets qui s'envoient des requêtes/messages.
- **Chaque objet à son propre espace mémoire** (il en est le propriétaire !), un objet pouvant contenir d'autres objets ;
- **Chaque objet est d'un type précis** et c'est son type (sa classe) qui caractérise les messages qu'il peut traiter ;
- **Des objets de types différents peuvent être d'un même type plus général** (*regroupant les caractéristiques et comportements en commun*) et comprendre les messages adressés pour le type général.

Les caractéristiques d'une approche POO pure pour la résolution d'un problème

- **Les objets sont des variables améliorées** qui reçoivent des requêtes/messages et se débrouillent pour les traiter ; *un message est un appel de fonction* ;
- Un programme est un ensemble d'objets qui s'envoient des requêtes/messages.
- **Chaque objet à son propre espace mémoire** (il en est le propriétaire !), un objet pouvant contenir d'autres objets ;
- **Chaque objet est d'un type précis** et c'est son type (sa classe) qui caractérise les messages qu'il peut traiter ;
- **Des objets de types différents peuvent être d'un même type plus général** (*regroupant les caractéristiques et comportements en commun*) et comprendre les messages adressés pour le type général.

Objectifs généraux

- **Sensibiliser les étudiants aux purs concepts de la POO** : objet, classe, constructeur, héritage, redéfinition, polymorphisme, programmation générique.
- **Sensibiliser les étudiants aux aspects spécifiques du Java utiles dans la conception d'application** : gestion d'exceptions, tests unitaires, les conteneurs.
- **Développer des compétences attractives pour les entreprises**

Objectifs généraux

- **Sensibiliser les étudiants aux purs concepts de la POO** : objet, classe, constructeur, héritage, redéfinition, polymorphisme, programmation générique.
- **Sensibiliser les étudiants aux aspects spécifiques du Java utiles dans la conception d'application** : gestion d'exceptions, tests unitaires, les conteneurs.
- **Développer des compétences attractives pour les entreprises**

Objectifs généraux

- **Sensibiliser les étudiants aux purs concepts de la POO** : objet, classe, constructeur, héritage, redéfinition, polymorphisme, programmation générique.
- **Sensibiliser les étudiants aux aspects spécifiques du Java utiles dans la conception d'application** : gestion d'exceptions, tests unitaires, les conteneurs.
- **Développer des compétences attractives pour les entreprises**

Objectifs spécifiques

A la fin du semestre, vous devriez être en mesure :

- de comprendre les intérêts de la Programmation Orientée Objet.
- d'utiliser à bon escient les classes abstraites, les interfaces et l'héritage simple.
- d'utiliser les types standard de Java, en particulier les String, les tableaux, les types énumérés et les collections.

Et pas seulement !

Objectifs spécifiques

A la fin du semestre, vous devriez être en mesure :

- de comprendre les intérêts de la Programmation Orientée Objet.
- d'utiliser à bon escient les classes abstraites, les interfaces et l'héritage simple.
- d'utiliser les types standard de Java, en particulier les String, les tableaux, les types énumérés et les collections.

Et pas seulement !

Objectifs spécifiques

A la fin du semestre, vous devriez être en mesure :

- de comprendre les intérêts de la Programmation Orientée Objet.
- d'utiliser à bon escient les classes abstraites, les interfaces et l'héritage simple.
- d'utiliser les types standard de Java, en particulier les String, les tableaux, les types énumérés et les collections.

Et pas seulement !

Fichier PremProg.java :

```
public class PremProg
{
   public static void main(String args[])
   {
      System.out.println( "Mon premier prog en Java" ) ;
   }
}
```

Fichier PremProg.java :

```
// public autorise les autres classes à y accèder (bonne habitude à prendre)
public class PremProg
{
   // public donne l'accès à la machine virtuelle
   // et permet ainsi l'exécution du programme
   public static void main(String args[]) // String args[] est obligatoire!
   {
      System.out.println( "Mon premier prog en Java" ) ; // affichage dans la fenêtre console
                                                         // avec changement de ligne
   }
   /* System est une classe
      System.out est un champ donnée de la classe System représentant une fenêtre console
      ; ce champ est static (static sera expliqué dans un autre cours)
      System.out.println est une méthode de l'objet out
      de type PrintStream */
}
```

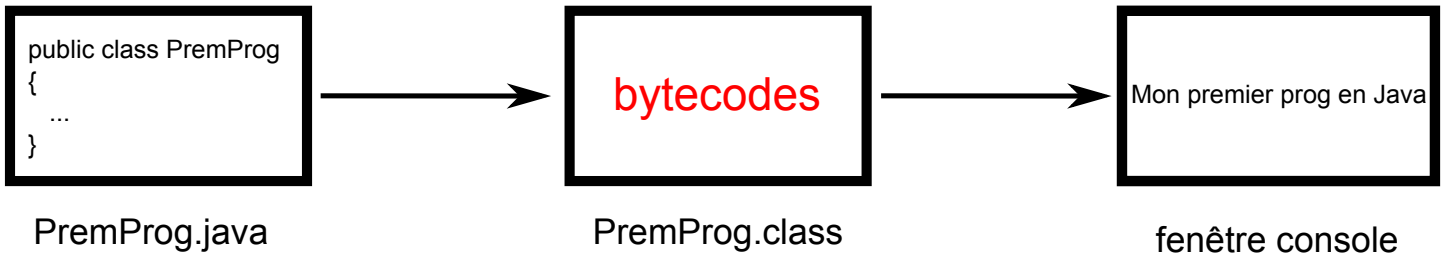
Il y a 2 formes de commentaires : ceux du C /* ... */ sur 1 ou plusieurs lignes, et les commentaires fin de ligne //.

Fichier PremProg.java :

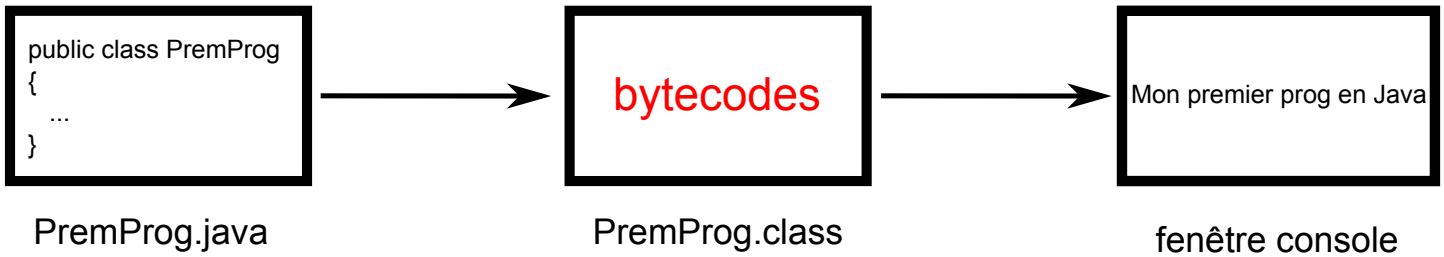
```
// public autorise les autres classes à y accèder (bonne habitude à prendre)
public class PremProg
{
   // public donne l'accès à la machine virtuelle
   // et permet ainsi l'exécution du programme
   public static void main(String args[]) // String args[] est obligatoire!
   {
      System.out.println( "Mon premier prog en Java" ) ; // affichage dans la fenêtre console
                                                         // avec changement de ligne
   }
   /* System est une classe
      System.out est un champ donnée de la classe System représentant une fenêtre console
      ; ce champ est static (static sera expliqué dans un autre cours)
      System.out.println est une méthode de l'objet out
      de type PrintStream */
}
```

Il y a 2 formes de commentaires : ceux du C /* ... */ sur 1 ou plusieurs lignes, et les commentaires fin de ligne //.

Le code source d'une *classe publique* DOIT TOUJOURS se trouver dans un fichier portant le même nom que la classe et possédant l'extension *java*.



Il faut parfois utiliser la commande java -classpath "." PremProg pour exécuter dans le répertoire contenant PremProg.class.



Il faut parfois utiliser la commande java -classpath "." PremProg pour exécuter dans le répertoire contenant PremProg.class.

- javac est un compilateur.
- java est un interpréteur de bytecode.

Sous Windows : Pensez à ajouter la variable d'environnement JAVA_HOME (à C :\Program Files\Java\jdk1.8.0_11 sur ma machine) et à rajouter %JAVA_HOME%\bin ; dans le Path (au début pour éviter les problèmes...).

- javac est un compilateur.
- java est un interpréteur de bytecode.

Sous Windows : Pensez à ajouter la variable d'environnement JAVA_HOME (à C :\Program Files\Java\jdk1.8.0_11 sur ma machine) et à rajouter %JAVA_HOME%\bin ; dans le Path (au début pour éviter les problèmes...).

- javac est un compilateur.
- java est un interpréteur de bytecode.

Sous Windows : Pensez à ajouter la variable d'environnement JAVA_HOME (à C :\Program Files\Java\jdk1.8.0_11 sur ma machine) et à rajouter %JAVA_HOME%\bin ; dans le Path (au début pour éviter les problèmes...).

Plan

Versions de Java

Java standard - Java SE

Nous utiliserons ponctuellement les fonctionnalités des versions 5 et 7 de Java SE.

Java EE (Enterprise Edition) et **Java ME** (Micro Edition)

Versions de Java

Java standard - Java SE

Nous utiliserons ponctuellement les fonctionnalités des versions 5 et 7 de Java SE.

Java EE (Enterprise Edition) et **Java ME** (Micro Edition)

Les outils

- Le **JRE** (*Java Runtime Environment*) contient la machine virtuelle Java nécessaire pour exécuter une application Java.
- Le **JDK** (*Java Development Kit*) comprend l'ensemble des outils pour développer une application Java :
  - Le JRE.
  - Le compilateur et des bibliothèques utilitaires.
  - Divers outils de développement tels que **javadoc** ou **jar**.

Installer le dernier JDK (*Java Development Kit*) standard de disponible : `http://www.oracle.com/technetwork/java/javase/downloads/index.html`

Les outils

- Le **JRE** (*Java Runtime Environment*) contient la machine virtuelle Java nécessaire pour exécuter une application Java.
- Le **JDK** (*Java Development Kit*) comprend l'ensemble des outils pour développer une application Java :
  - Le JRE.
  - Le compilateur et des bibliothèques utilitaires.
  - Divers outils de développement tels que **javadoc** ou **jar**.

Installer le dernier JDK (*Java Development Kit*) standard de disponible : `http://www.oracle.com/technetwork/java/javase/downloads/index.html`

Les outils

- Le **JRE** (*Java Runtime Environment*) contient la machine virtuelle Java nécessaire pour exécuter une application Java.
- Le **JDK** (*Java Development Kit*) comprend l'ensemble des outils pour développer une application Java :
  - Le JRE.
  - Le compilateur et des bibliothèques utilitaires.
  - Divers outils de développement tels que **javadoc** ou **jar**.

Installer le dernier JDK (*Java Development Kit*) standard de disponible : `http://www.oracle.com/technetwork/java/javase/downloads/index.html`

Les outils

- Le **JRE** (*Java Runtime Environment*) contient la machine virtuelle Java nécessaire pour exécuter une application Java.
- Le **JDK** (*Java Development Kit*) comprend l'ensemble des outils pour développer une application Java :
  - Le JRE.
  - Le compilateur et des bibliothèques utilitaires.
  - Divers outils de développement tels que **javadoc** ou **jar**.

Installer le dernier JDK (*Java Development Kit*) standard de disponible : `http://www.oracle.com/technetwork/java/javase/downloads/index.html`

Les outils

- Le **JRE** (*Java Runtime Environment*) contient la machine virtuelle Java nécessaire pour exécuter une application Java.
- Le **JDK** (*Java Development Kit*) comprend l'ensemble des outils pour développer une application Java :
  - Le JRE.
  - Le compilateur et des bibliothèques utilitaires.
  - Divers outils de développement tels que **javadoc** ou **jar**.

Installer le dernier JDK (*Java Development Kit*) standard de disponible : `http://www.oracle.com/technetwork/java/javase/downloads/index.html`

La documentation des classes standards Java et de leurs méthodes

La documentation officielle est disponible ici : `http://www.oracle.com/technetwork/java/index.html`

L'utilitaire jar (Java Archive)

- Rapidement beaucoup de classes à compiler.
- Des fichiers liés à l'application Java : .properties, .xml etc.
- => créer un **JAR** (*Java **AR**chive*) ou un **WAR** (*Web **AR**chive*) pour mettre ensemble des classes Java et des ressources (images, etc.) dans un fichier à distribuer/à déployer.
  Une archive est soit une bibliothèque, soit une application stand-alone (une application à part entière).
- On peut rendre les **JAR** auto-exécutables : `http://openclassrooms.com/courses/creer-une-archive-jar`

L'utilitaire jar (Java Archive)

- Rapidement beaucoup de classes à compiler.
- Des fichiers liés à l'application Java : .properties, .xml etc.
- => créer un **JAR** (*Java **AR**chive*) ou un **WAR** (*Web **AR**chive*) pour mettre ensemble des classes Java et des ressources (images, etc.) dans un fichier à distribuer/à déployer.
  Une archive est soit une bibliothèque, soit une application stand-alone (une application à part entière).
- On peut rendre les **JAR** auto-exécutables : `http://openclassrooms.com/courses/creer-une-archive-jar`

L'utilitaire jar (Java Archive)

- Rapidement beaucoup de classes à compiler.
- Des fichiers liés à l'application Java : .properties, .xml etc.
- => créer un **JAR** (***J**ava **AR**chive*) ou un **WAR** (***W**eb **AR**chive*) pour mettre ensemble des classes Java et des ressources (images, etc.) dans un fichier à distribuer/à déployer.
  Une archive est soit une bibliothèque, soit une application stand-alone (une application à part entière).
- On peut rendre les **JAR** auto-exécutables : http://openclassrooms.com/courses/creer-une-archive-jar

L'utilitaire jar (Java Archive)

- Rapidement beaucoup de classes à compiler.
- Des fichiers liés à l'application Java : .properties, .xml etc.
- => créer un **JAR** (***J**ava **AR**chive*) ou un **WAR** (***W**eb **AR**chive*) pour mettre ensemble des classes Java et des ressources (images, etc.) dans un fichier à distribuer/à déployer.
  Une archive est soit une bibliothèque, soit une application stand-alone (une application à part entière).
- On peut rendre les **JAR** auto-exécutables :
  `http://openclassrooms.com/courses/creer-une-archive-jar`

Gestionnaire de version préconisé : Git

Mettre votre code sur un serveur et :

- synchroniser votre travail sur plusieurs machines et avec votre binôme.
- suivi des modifications, revenir à une version précédente, tester plusieurs solutions dans plusieurs branches et ne conserver à la fin que la meilleure solution etc.

Plus d'information dans le document gestion_des_versions_de_vos_projets disponible sur spiralconnect.

Obligatoire d'utiliser un gestionnaire de version **privé** pour vos TPs évalués.

Gestionnaire de version préconisé : Git

Mettre votre code sur un serveur et :

- synchroniser votre travail sur plusieurs machines et avec votre binôme.
- suivi des modifications, revenir à une version précédente, tester plusieurs solutions dans plusieurs branches et ne conserver à la fin que la meilleure solution etc.

Plus d'information dans le document gestion_des_versions_de_vos_projets disponible sur spiralconnect.

Obligatoire d'utiliser un gestionnaire de version **privé** pour vos TPs évalués.

Gestionnaire de version préconisé : Git

Mettre votre code sur un serveur et :

- synchroniser votre travail sur plusieurs machines et avec votre binôme.
- suivi des modifications, revenir à une version précédente, tester plusieurs solutions dans plusieurs branches et ne conserver à la fin que la meilleure solution etc.

Plus d'information dans le document gestion_des_versions_de_vos_projets disponible sur spiralconnect.

Obligatoire d'utiliser un gestionnaire de version **privé** pour vos TPs évalués.

Gestionnaire de version préconisé : Git

Mettre votre code sur un serveur et :

- synchroniser votre travail sur plusieurs machines et avec votre binôme.
- suivi des modifications, revenir à une version précédente, tester plusieurs solutions dans plusieurs branches et ne conserver à la fin que la meilleure solution etc.

Plus d'information dans le document gestion_des_versions_de_vos_projets disponible sur spiralconnect.

Obligatoire d'utiliser un gestionnaire de version **privé** pour vos TPs évalués.

Exécution pas à pas et en ligne de code Java

http ://www.pythontutor.com/ :

`http://www.pythontutor.com/visualize.html#mode=edit`

Environnement de Développement Intégré - EDI

Les EDIs les plus utilisés en Java :

- NetBeans : `https://netbeans.org/` ;
- eclipse : `http://www.eclipse.org/` ;
- IntelliJ : `http://www.jetbrains.com/idea/`.

Ils sont pratiques pour développer, mais pas d'automatisation directe de la gestion des dépendances pour un projet propre à l'EDI.
C'est en fait Ant ou Maven intégrés à vos EDI qui vont gérer automatiquement les dépendances pour vous.

Environnement de Développement Intégré - EDI

Les EDIs les plus utilisés en Java :

- NetBeans : `https://netbeans.org/` ;
- eclipse : `http://www.eclipse.org/` ;
- IntelliJ : `http://www.jetbrains.com/idea/`.

Ils sont pratiques pour développer, mais pas d'automatisation directe de la gestion des dépendances pour un projet propre à l'EDI.
C'est en fait Ant ou Maven intégrés à vos EDI qui vont gérer automatiquement les dépendances pour vous.

Environnement de Développement Intégré - EDI

Les EDIs les plus utilisés en Java :

- NetBeans : `https://netbeans.org/` ;
- eclipse : `http://www.eclipse.org/` ;
- IntelliJ : `http://www.jetbrains.com/idea/`.

Ils sont pratiques pour développer, mais pas d'automatisation directe de la gestion des dépendances pour un projet propre à l'EDI.
C'est en fait Ant ou Maven intégrés à vos EDI qui vont gérer automatiquement les dépendances pour vous.

Environnement de Développement Intégré - EDI

Les EDIs les plus utilisés en Java :

- NetBeans : `https://netbeans.org/` ;
- eclipse : `http://www.eclipse.org/` ;
- IntelliJ : `http://www.jetbrains.com/idea/`.

Ils sont pratiques pour développer, mais pas d'automatisation directe de la gestion des dépendances pour un projet propre à l'EDI.
C'est en fait Ant ou Maven intégrés à vos EDI qui vont gérer automatiquement les dépendances pour vous.

Environnement de Développement Intégré - EDI

Les EDIs les plus utilisés en Java :

- NetBeans : `https://netbeans.org/` ;
- eclipse : `http://www.eclipse.org/` ;
- IntelliJ : `http://www.jetbrains.com/idea/`.

Ils sont pratiques pour développer, mais pas d'automatisation directe de la gestion des dépendances pour un projet propre à l'EDI.

C'est en fait Ant ou Maven intégrés à vos EDI qui vont gérer automatiquement les dépendances pour vous.

Environnement de Développement Intégré - EDI

Les EDIs les plus utilisés en Java :

- NetBeans : `https://netbeans.org/` ;
- eclipse : `http://www.eclipse.org/` ;
- IntelliJ : `http://www.jetbrains.com/idea/`.

Ils sont pratiques pour développer, mais pas d'automatisation directe de la gestion des dépendances pour un projet propre à l'EDI.
C'est en fait Ant ou Maven intégrés à vos EDI qui vont gérer automatiquement les dépendances pour vous.

Maven

On pourra utiliser Maven pour créer le projet initial. On peut créer un projet Maven via *NetBeans*, *eclipse IDE for Java developers* et *IntelliJ*.

Le fichier POM (projet multi-modules) du projet Maven permettra d'automatiser la gestion des dépendances via le *repository* central de Maven dans les différents *scope* (compile, runtime, test). Très utile si on travaille à plusieurs sur 1 projet.

Alternatives à Maven : **Ant** mais surtout **GRADLE** : `http://www.gradle.org/` avec son plugin Java `http://www.gradle.org/docs/current/userguide/java_plugin.html`.

Maven

On pourra utiliser Maven pour créer le projet initial. On peut créer un projet Maven via *NetBeans*, *eclipse IDE for Java developers* et *IntelliJ*.

Le fichier POM (projet multi-modules) du projet Maven permettra d'automatiser la gestion des dépendances via le *repository* central de Maven dans les différents *scope* (compile, runtime, test). Très utile si on travaille à plusieurs sur 1 projet.

Alternatives à Maven : **Ant** mais surtout **GRADLE** : `http://www.gradle.org/` avec son plugin Java `http://www.gradle.org/docs/current/userguide/java_plugin.html`.

Maven

On pourra utiliser Maven pour créer le projet initial. On peut créer un projet Maven via *NetBeans*, *eclipse IDE for Java developers* et *IntelliJ*.

Le fichier POM (projet multi-modules) du projet Maven permettra d'automatiser la gestion des dépendances via le *repository* central de Maven dans les différents *scope* (compile, runtime, test). Très utile si on travaille à plusieurs sur 1 projet.

Alternatives à Maven : **Ant** mais surtout **GRADLE** : `http://www.gradle.org/` avec son plugin Java `http://www.gradle.org/docs/current/userguide/java_plugin.html`.

Maven

On pourra utiliser Maven pour créer le projet initial. On peut créer un projet Maven via *NetBeans*, *eclipse IDE for Java developers* et *IntelliJ*.

Le fichier POM (projet multi-modules) du projet Maven permettra d'automatiser la gestion des dépendances via le *repository* central de Maven dans les différents *scope* (compile, runtime, test). Très utile si on travaille à plusieurs sur 1 projet.

Alternatives à Maven : **Ant** mais surtout **GRADLE** : `http://www.gradle.org/` avec son plugin Java `http://www.gradle.org/docs/current/userguide/java_plugin.html`.

Fichier POM de maven

- <groupId>fr.iut.lyon1.etu</groupId>
- <artifactId>nom-programme</artifactId> ; *nom-programme = nom de la classe contenant la méthode main*
- <version>0.1-SNAPSHOT</version>
- <packaging>jar</packaging>
- <dependencies>
  - <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.10</version>
    <scope>test</scope>
    </dependency>
- </dependencies>

Fichier POM de maven

- <groupId>fr.iut.lyon1.etu</groupId>
- <artifactId>nom-programme</artifactId> ; *nom-programme = nom de la classe contenant la méthode main*
- <version>0.1-SNAPSHOT</version>
- <packaging>jar</packaging>
- <dependencies>
  - <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.10</version>
    <scope>test</scope>
    </dependency>
- </dependencies>

Fichier POM de maven

- <groupId>fr.iut.lyon1.etu</groupId>
- <artifactId>nom-programme</artifactId> ; *nom-programme = nom de la classe contenant la méthode main*
- <version>0.1-SNAPSHOT</version>
- <packaging>jar</packaging>
- <dependencies>
  - <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.10</version>
    <scope>test</scope>
    </dependency>
- </dependencies>

Plan

3 formes de commentaires en Java :

- /* Sur plusieurs
  lignes */
- // fin de ligne
- Commentaire pour la génération automatique de documentation avec l'outil **JavaDoc** :
  /** commentaire utile
  aux utilisateurs de votre API */

3 formes de commentaires en Java :

- /* Sur plusieurs
  lignes */
- // fin de ligne
- Commentaire pour la génération automatique de documentation avec l'outil **JavaDoc** :
  /** commentaire utile
  aux utilisateurs de votre API */

3 formes de commentaires en Java :

- /* Sur plusieurs
  lignes */
- // fin de ligne
- Commentaire pour la génération automatique de documentation avec l'outil **JavaDoc** :
  /** commentaire utile
  aux utilisateurs de votre API */

Déclaration des variables obligatoire, mais pas nécessairement en début de block { ... }.

Exemple.java

```
public class Exemple
{ public static void main (String[] args)
  { int n ;    // valeur de n aléatoire
    double x ; // valeur de x aléatoire
    n = 5 ;
    x = 2*n + 1.5 ;
    System.out.println ("n = " + n) ;
    System.out.println ("x = " + x) ;
    double y ;
    y = n * x + 12 ;
    System.out.println ("y = " + y) ;
  }
}
```

+ : dès qu'un de ses 2 opérandes est une chaîne, l'autre est converti en chaîne !

Attention : une variable utilisée, mais non initialisée engendre une erreur de compilation.

Déclaration des variables obligatoire, mais pas nécessairement en début de block { ... }.

Exemple.java

```
public class Exemple
{ public static void main (String[] args)
  { int n ;    // valeur de n aléatoire
    double x ; // valeur de x aléatoire
    n = 5 ;
    x = 2*n + 1.5 ;
    System.out.println ("n = " + n) ;
    System.out.println ("x = " + x) ;
    double y ;
    y = n * x + 12 ;
    System.out.println ("y = " + y) ;
  }
}
```

+ : dès qu'un de ses 2 opérandes est une chaîne, l'autre est converti en chaîne !

Attention : une variable utilisée, mais non initialisée engendre une erreur de compilation.

Déclaration des variables obligatoire, mais pas nécessairement en début de block { ... }.

Exemple.java

```
public class Exemple
{ public static void main (String[] args)
  { int n ;    // valeur de n aléatoire
    double x ; // valeur de x aléatoire
    n = 5 ;
    x = 2*n + 1.5 ;
    System.out.println ("n = " + n) ;
    System.out.println ("x = " + x) ;
    double y ;
    y = n * x + 12 ;
    System.out.println ("y = " + y) ;
  }
}
```

+ : dès qu'un de ses 2 opérandes est une chaîne, l'autre est converti en chaîne !

Attention : une variable utilisée, mais non initialisée engendre une erreur de compilation.

Java refuse les conversions implicites qui peuvent dégrader les données : double vers float, float vers long et long vers int.

Exemple2.java

```
public class Exemple2
{ public static void main (String[] args)
  { int n ;
    float x ;
    n = 5 ;
    x = 2*n + 1.5 ;
    System.out.println ("n = " + n) ;
    System.out.println ("x = " + x) ;
    double y ;
    y = n * x + 12 ;
    System.out.println ("y = " + y) ;
  }
}
```

NE COMPILE PAS !

Java refuse les conversions implicites qui peuvent dégrader les données : double vers float, float vers long et long vers int.

Exemple2.java

```
public class Exemple2
{ public static void main (String[] args)
  { int n ;
    float x ;
    n = 5 ;
    x = 2*n + 1.5 ;
    System.out.println ("n = " + n) ;
    System.out.println ("x = " + x) ;
    double y ;
    y = n * x + 12 ;
    System.out.println ("y = " + y) ;
  }
}
```

NE COMPILE PAS !

Java refuse les conversions implicites qui peuvent dégrader les données : double vers float, float vers long et long vers int.

Exemple2.java

```
public class Exemple2
{ public static void main (String[] args)
  { int n ;
    float x ;
    n = 5 ;
    x = 2*n + 1.5 ;
    System.out.println ("n = " + n) ;
    System.out.println ("x = " + x) ;
    double y ;
    y = n * x + 12 ;
    System.out.println ("y = " + y) ;
  }
}
```

NE COMPILE PAS !

Solution : 1.5**f**

Nommage en Java (règles communes)

- **1 seule langue** (anglais ou français).
- Des **noms simples et représentatifs** sans ambiguïté.
- Ne pas utiliser les caractères : -, les symboles ($, @, ...), les caractères accentués.
- **1 convention fixe pour la casse et la séparation des mots** :
  - Les variables, objets et fonctions : tout en minuscule et la première lettre de chaque mot *sauf le 1er* en majuscule.
  - Les classes : tout en minuscule et la première lettre de chaque mot en majuscule.
  - Les constantes : tout en majuscule et séparation des mots par _.

Dans System.out.println, System est une classe.

Nommage en Java (règles communes)

- **1 seule langue** (anglais ou français).
- Des **noms simples et représentatifs** sans ambiguïté.
- Ne pas utiliser les caractères : -, les symboles ($, @, ...), les caractères accentués.
- **1 convention fixe pour la casse et la séparation des mots** :
  - Les variables, objets et fonctions : tout en minuscule et la première lettre de chaque mot *sauf le 1er* en majuscule.
  - Les classes : tout en minuscule et la première lettre de chaque mot en majuscule.
  - Les constantes : tout en majuscule et séparation des mots par _.

Dans System.out.println, System est une classe.

Nommage en Java (règles communes)

- **1 seule langue** (anglais ou français).
- Des **noms simples et représentatifs** sans ambiguïté.
- Ne pas utiliser les caractères : -, les symboles ($, @, ...), les caractères accentués.
- **1 convention fixe pour la casse et la séparation des mots** :
  - Les variables, objets et fonctions : tout en minuscule et la première lettre de chaque mot *sauf le 1er* en majuscule.
  - Les classes : tout en minuscule et la première lettre de chaque mot en majuscule.
  - Les constantes : tout en majuscule et séparation des mots par _.

Dans System.out.println, System est une classe.

Nommage en Java (règles communes)

- **1 seule langue** (anglais ou français).
- Des **noms simples et représentatifs** sans ambiguïté.
- Ne pas utiliser les caractères : -, les symboles ($, @, ...), les caractères accentués.
- **1 convention fixe pour la casse et la séparation des mots** :
  - Les variables, objets et fonctions : tout en minuscule et la première lettre de chaque mot *sauf le 1er* en majuscule.
  - Les classes : tout en minuscule et la première lettre de chaque mot en majuscule.
  - Les constantes : tout en majuscule et séparation des mots par _.

Dans System.out.println, System est une classe.

Nommage en Java (règles communes)

- **1 seule langue** (anglais ou français).
- Des **noms simples et représentatifs** sans ambiguïté.
- Ne pas utiliser les caractères : -, les symboles ($, @, ...), les caractères accentués.
- **1 convention fixe pour la casse et la séparation des mots** :
  - Les variables, objets et fonctions : tout en minuscule et la première lettre de chaque mot *sauf le 1er* en majuscule.
  - Les classes : tout en minuscule et la première lettre de chaque mot en majuscule.
  - Les constantes : tout en majuscule et séparation des mots par _.

Dans System.out.println, System est une classe.

Nommage en Java (règles communes)

- **1 seule langue** (anglais ou français).
- Des **noms simples et représentatifs** sans ambiguïté.
- Ne pas utiliser les caractères : -, les symboles ($, @, ...), les caractères accentués.
- **1 convention fixe pour la casse et la séparation des mots** :
  - Les variables, objets et fonctions : tout en minuscule et la première lettre de chaque mot *sauf le 1er* en majuscule.
  - Les classes : tout en minuscule et la première lettre de chaque mot en majuscule.
  - Les constantes : tout en majuscule et séparation des mots par _.

Dans System.out.println, System est une classe.

Nommage en Java (règles communes)

- **1 seule langue** (anglais ou français).
- Des **noms simples et représentatifs** sans ambiguïté.
- Ne pas utiliser les caractères : -, les symboles ($, @, ...), les caractères accentués.
- **1 convention fixe pour la casse et la séparation des mots** :
  - Les variables, objets et fonctions : tout en minuscule et la première lettre de chaque mot *sauf le 1er* en majuscule.
  - Les classes : tout en minuscule et la première lettre de chaque mot en majuscule.
  - Les constantes : tout en majuscule et séparation des mots par _.

Dans System.out.println, System est une classe.

Java permet un affichage dans la console via System.out.println.
Il n'existe rien de comparable pour la lecture clavier.

On vous fournit sur spiralconnect une classe publique pour réaliser des lectures clavier : Clavier dans le fichier Clavier.java.
Elle donne accès aux méthodes statiques suivantes :

- public static String lireString () { ... }
- public static float lireFloat () { ... }
- public static double lireDouble () { ... }
- public static int lireInt () { ... }
- public static char lireChar () { ... }

Pour le moment il est encore trop tôt pour comprendre le corps de ces méthodes, contentez-vous de les utiliser.

Java permet un affichage dans la console via System.out.println.
Il n'existe rien de comparable pour la lecture clavier.
On vous fournit sur spiralconnect une classe publique pour réaliser des lectures clavier : Clavier dans le fichier Clavier.java.
Elle donne accès aux méthodes statiques suivantes :

- public static String lireString () { ... }
- public static float lireFloat () { ... }
- public static double lireDouble () { ... }
- public static int lireInt () { ... }
- public static char lireChar () { ... }

Pour le moment il est encore trop tôt pour comprendre le corps de ces méthodes, contentez-vous de les utiliser.

Java permet un affichage dans la console via System.out.println.
Il n'existe rien de comparable pour la lecture clavier.
On vous fournit sur spiralconnect une classe publique pour réaliser des lectures clavier : Clavier dans le fichier Clavier.java.
Elle donne accès aux méthodes statiques suivantes :

- public static String lireString () { ... }
- public static float lireFloat () { ... }
- public static double lireDouble () { ... }
- public static int lireInt () { ... }
- public static char lireChar () { ... }

Pour le moment il est encore trop tôt pour comprendre le corps de ces méthodes, contentez-vous de les utiliser.

Java permet un affichage dans la console via System.out.println.
Il n'existe rien de comparable pour la lecture clavier.
On vous fournit sur spiralconnect une classe publique pour réaliser des lectures clavier : Clavier dans le fichier Clavier.java.
Elle donne accès aux méthodes statiques suivantes :

- public static String lireString () { ... }
- public static float lireFloat () { ... }
- public static double lireDouble () { ... }
- public static int lireInt () { ... }
- public static char lireChar () { ... }

Pour le moment il est encore trop tôt pour comprendre le corps de ces méthodes, contentez-vous de les utiliser.

Java permet un affichage dans la console via System.out.println.
Il n'existe rien de comparable pour la lecture clavier.
On vous fournit sur spiralconnect une classe publique pour réaliser des lectures clavier : Clavier dans le fichier Clavier.java.
Elle donne accès aux méthodes statiques suivantes :

- public static String lireString () { ... }
- public static float lireFloat () { ... }
- public static double lireDouble () { ... }
- public static int lireInt () { ... }
- public static char lireChar () { ... }

Pour le moment il est encore trop tôt pour comprendre le corps de ces méthodes, contentez-vous de les utiliser.

Java permet un affichage dans la console via System.out.println.
Il n'existe rien de comparable pour la lecture clavier.
On vous fournit sur spiralconnect une classe publique pour réaliser des lectures clavier : Clavier dans le fichier Clavier.java.
Elle donne accès aux méthodes statiques suivantes :

- public static String lireString () { ... }
- public static float lireFloat () { ... }
- public static double lireDouble () { ... }
- public static int lireInt () { ... }
- public static char lireChar () { ... }

Pour le moment il est encore trop tôt pour comprendre le corps de ces méthodes, contentez-vous de les utiliser.

Java permet un affichage dans la console via System.out.println.
Il n'existe rien de comparable pour la lecture clavier.
On vous fournit sur spiralconnect une classe publique pour réaliser des lectures clavier : Clavier dans le fichier Clavier.java.
Elle donne accès aux méthodes statiques suivantes :

- public static String lireString () { ... }
- public static float lireFloat () { ... }
- public static double lireDouble () { ... }
- public static int lireInt () { ... }
- public static char lireChar () { ... }

Pour le moment il est encore trop tôt pour comprendre le corps de ces méthodes, contentez-vous de les utiliser.

```
class SaisieClavier {
    public static void main (String[] arg) {

       System.out.println("Donnez votre prénom et votre nom") ;
       String prenom = Clavier.lireString() ;
       String nom = Clavier.lireString() ;
       System.out.println("Donnez votre âge") ;
       int age = Clavier.lireInt() ;

       System.out.println("Ecrire votre phrase") ;
       String phrase = Clavier.lireString() ;
       System.out.println(prenom + " " + nom + ", " +
                          age + " ans, dit : " + phrase) ;

    }
}
```

Autre méthode : utiliser un Scanner

Depuis le JDK 5.0, il est possible de faire une saisie clavier à partir de la classe java.util.Scanner.

```
class SaisieClavierBis {
    public static void main (String[] arg) {

        java.util.Scanner entree = new java.util.Scanner(System.in) ;

        System.out.println("Donnez votre prénom et votre nom") ;
        String prenom = entree.next() ;
        String nom = entree.next() ;
        System.out.println("Donnez votre âge") ;
        int age = entree.nextInt() ;
        entree.nextLine() ;
        System.out.println("Ecrire votre phrase") ;
        String phrase = entree.nextLine() ;
        System.out.println(prenom + " " + nom + ", " +
                           age + " ans, dit : " + phrase) ;

    }
}
```

Elles sont semblables à celles du C/C++ (for, while, do...while, if, if-else, switch). *La seule différence est l'existence en Java d'instruction break et continue avec une étiquette.*

Racines.java

```
// La classe Racines utilise la classe Clavier
public class Racines
{ public static void main (String[] args)
  { final int NFOIS = 5 ; // NFOIS ne peut pas changer de valeur dans la suite
    int i ;
    double x ;
    double racx ;

    System.out.println ("Bonjour") ;
    System.out.println ("Je vais vous calculer " + NFOIS + " racines carrees") ;

    for (i=0 ; i<NFOIS ; i++)
      { System.out.print ("Donnez un nombre : ") ;
        x  = Clavier.lireDouble ()  ;
        if (x < 0.0)
          System.out.println (x + " ne possede pas de racine carree") ;
        else
         { racx = Math.sqrt(x) ;
           System.out.println (x + " a pour racine carree : "  + racx) ;
         }
      }
    System.out.println ("Travail termine - Au revoir") ;
  }
}
```

Elles sont semblables à celles du C/C++ (for, while, do...while, if, if-else, switch). *La seule différence est l'existence en Java d'instruction break et continue avec une étiquette*.

Racines.java

```
// La classe Racines utilise la classe Clavier
public class Racines
{ public static void main (String[] args)
  { final int NFOIS = 5 ; // NFOIS ne peut pas changer de valeur dans la suite
    int i ;
    double x ;
    double racx ;

    System.out.println ("Bonjour") ;
    System.out.println ("Je vais vous calculer " + NFOIS + " racines carrees") ;

    for (i=0 ; i<NFOIS ; i++)
      { System.out.print ("Donnez un nombre : ") ;
        x  = Clavier.lireDouble ()  ;
        if (x < 0.0)
          System.out.println (x + " ne possede pas de racine carree") ;
         else
          { racx = Math.sqrt(x) ;
            System.out.println (x + " a pour racine carree : "  + racx) ;
          }
      }
    System.out.println ("Travail termine - Au revoir") ;
  }
}
```

Utilisation la plus courante du break avec étiquette :

```
etiq : while(true){
   ...
   while(...){
      ...
      if(...) break etiq ; //  Arrête l'itération étiquettée par etiq
      ...
    }
    ...
}
```

Une nouvelle syntaxe possible pour le for à partir du JDK 5.0 :

```
for([type élément] <id-var> : <expression>) <instruction> ;
```

Expression doit être un tableau ou une instance d'une classe implémentant l'interface Iterable. On verra cela plus tard.

Ce sont les seuls types du langage qui ne sont pas des classes. Ils seront utilisés principalement pour définir certains des champs des classes.

4 catégories de type primitif :

- **Les nombres entiers** codés en C2 : byte sur 8 bits, short sur 16 bits, int sur 32 bits et long sur 64 bits.
- **Les nombres flottants** codés en IEEE 754 : float sur 32 bits (7 chiffres significatifs) et double sur 64 bits (15 chiffres significatifs).
- **Les caractères** : char sur 16 bits. *Pourquoi 16 bits ?*
- **Les booléens** : boolean sur 8 bits (dépend de si variable ou tableau).

Type.MIN_VALUE et Type.MAX_VALUE sont des constantes parfois utiles.

Ce sont les seuls types du langage qui ne sont pas des classes. Ils seront utilisés principalement pour définir certains des champs des classes.

4 catégories de type primitif :

- **Les nombres entiers** codés en C2 : `byte` sur 8 bits, `short` sur 16 bits, `int` sur 32 bits et `long` sur 64 bits.
- **Les nombres flottants** codés en IEEE 754 : `float` sur 32 bits (7 chiffres significatifs) et `double` sur 64 bits (15 chiffres significatifs).
- **Les caractères** : `char` sur 16 bits. *Pourquoi 16 bits ?*
- **Les booléens** : `boolean` sur 8 bits (dépend de si variable ou tableau).

`Type.MIN_VALUE` et `Type.MAX_VALUE` sont des constantes parfois utiles.

Ce sont les seuls types du langage qui ne sont pas des classes. Ils seront utilisés principalement pour définir certains des champs des classes.

4 catégories de type primitif :

- **Les nombres entiers** codés en C2 : byte sur 8 bits, short sur 16 bits, int sur 32 bits et long sur 64 bits.
- **Les nombres flottants** codés en IEEE 754 : float sur 32 bits (7 chiffres significatifs) et double sur 64 bits (15 chiffres significatifs).
- **Les caractères** : char sur 16 bits. *Pourquoi 16 bits ?*
- **Les booléens** : boolean sur 8 bits (dépend de si variable ou tableau).

Type.MIN_VALUE et Type.MAX_VALUE sont des constantes parfois utiles.

Ce sont les seuls types du langage qui ne sont pas des classes. Ils seront utilisés principalement pour définir certains des champs des classes.

4 catégories de type primitif :

- **Les nombres entiers** codés en C2 : byte sur 8 bits, short sur 16 bits, int sur 32 bits et long sur 64 bits.
- **Les nombres flottants** codés en IEEE 754 : float sur 32 bits (7 chiffres significatifs) et double sur 64 bits (15 chiffres significatifs).
- **Les caractères** : char sur 16 bits. *Pourquoi 16 bits ?*
- **Les booléens** : boolean sur 8 bits (dépend de si variable ou tableau).

Type.MIN_VALUE et Type.MAX_VALUE sont des constantes parfois utiles.

Ce sont les seuls types du langage qui ne sont pas des classes. Ils seront utilisés principalement pour définir certains des champs des classes.

4 catégories de type primitif :

- **Les nombres entiers** codés en C2 : byte sur 8 bits, short sur 16 bits, int sur 32 bits et long sur 64 bits.
- **Les nombres flottants** codés en IEEE 754 : float sur 32 bits (7 chiffres significatifs) et double sur 64 bits (15 chiffres significatifs).
- **Les caractères** : char sur 16 bits. *Pourquoi 16 bits ?*
- **Les booléens** : boolean sur 8 bits (dépend de si variable ou tableau).

Type.MIN_VALUE et Type.MAX_VALUE sont des constantes parfois utiles.

Ce sont les seuls types du langage qui ne sont pas des classes. Ils seront utilisés principalement pour définir certains des champs des classes.

4 catégories de type primitif :

- **Les nombres entiers** codés en C2 : byte sur 8 bits, short sur 16 bits, int sur 32 bits et long sur 64 bits.
- **Les nombres flottants** codés en IEEE 754 : float sur 32 bits (7 chiffres significatifs) et double sur 64 bits (15 chiffres significatifs).
- **Les caractères** : char sur 16 bits. *Pourquoi 16 bits ?*
- **Les booléens** : boolean sur 8 bits (dépend de si variable ou tableau).

Type.MIN_VALUE et Type.MAX_VALUE sont des constantes parfois utiles.

Ce sont les seuls types du langage qui ne sont pas des classes. Ils seront utilisés principalement pour définir certains des champs des classes.

4 catégories de type primitif :

- **Les nombres entiers** codés en C2 : byte sur 8 bits, short sur 16 bits, int sur 32 bits et long sur 64 bits.
- **Les nombres flottants** codés en IEEE 754 : float sur 32 bits (7 chiffres significatifs) et double sur 64 bits (15 chiffres significatifs).
- **Les caractères** : char sur 16 bits. *Pourquoi 16 bits ?*
- **Les booléens** : boolean sur 8 bits (dépend de si variable ou tableau).

Type.MIN_VALUE et Type.MAX_VALUE sont des constantes parfois utiles.

Notation pour les constantes numériques

On utilise les mêmes notations pour les constantes qu'en C/C++.

`12.f` : le `f` ou `F` pour obtenir une constante de type `float` vous sera utile, car `float x = 12.;` ne compilera pas !

Pour les caractères, une nouveauté : on peut représenter un caractère par son code unicode de la forme `\uxxxx`.

Notation pour les constantes numériques

On utilise les mêmes notations pour les constantes qu'en C/C++.

$12.\mathrm{f}$: le f ou F pour obtenir une constante de type float vous sera utile, car $\mathrm{float\ x} = 12.$; ne compilera pas !

Pour les caractères, une nouveauté : on peut représenter un caractère par son code unicode de la forme $\backslash uxxxx$.

Notation pour les constantes numériques

On utilise les mêmes notations pour les constantes qu'en C/C++.

`12.f` : le `f` ou `F` pour obtenir une constante de type `float` vous sera utile, car `float x = 12.;` ne compilera pas !

Pour les caractères, une nouveauté : on peut représenter un caractère par son code unicode de la forme `\uxxxx`.

Le mot-clé final

Java permet de déclarer que la valeur d'une variable ne doit pas être modifiée pendant l'exécution du programme *suite à sa 1ère affectation*.

Exemples :

```
final int n = 20 ;

int p = 5 ;
final int m = 2*p - 3 ;

final int o ;
...
o = 4 ;
```

Le mot-clé final

Java permet de déclarer que la valeur d'une variable ne doit pas être modifiée pendant l'exécution du programme *suite à sa 1ère affectation*.

Exemples :

```
final int n = 20 ;

int p = 5 ;
final int m = 2*p - 3 ;

final int o ;
...
o = 4 ;
```

Le mot-clé final

Java permet de déclarer que la valeur d'une variable ne doit pas être modifiée pendant l'exécution du programme *suite à sa 1ère affectation*.

Exemples :

```
final int n = 20 ;

int p = 5 ;
final int m = 2*p - 3 ;

final int o ;
...
o = 4 ;
```

On retrouve les mêmes opérateurs qu'en C/C++ (même l'opérateur de cast), avec les différences suivantes :

- L'opérateur % fonctionne également pour les entiers négatifs et les flottants.
- La division par zéro pour les entiers génère une ArithmeticException ; si l'exception n'est pas interceptée, le programme s'arrête.
- Les flottants respectent les conventions IEEE 754 : il y a une représentation de l'infini positif, de l'infini négatif et d'une valeur non calculable (NaN) ; 1. / 0. donnera l'infini positif ! On peut accéder à ces représentations particulières, e.g. Float.POSITIVE_INFINITY. *Attention Float ≠ float !*
- Les opérateurs numériques (+, -, *, / etc.) ne sont pas définis pour les types byte, char et short. **Java met en place une promotion numérique vers le type** int.

On retrouve les mêmes opérateurs qu'en C/C++ (même l'opérateur de cast), avec les différences suivantes :

- L'opérateur % fonctionne également pour les entiers négatifs et les flottants.
- La division par zéro pour les entiers génère une ArithmeticException ; si l'exception n'est pas interceptée, le programme s'arrête.
- Les flottants respectent les conventions IEEE 754 : il y a une représentation de l'infini positif, de l'infini négatif et d'une valeur non calculable (NaN) ; 1. / 0. donnera l'infini positif ! On peut accéder à ces représentations particulières, e.g. Float.POSITIVE_INFINITY. *Attention Float ≠ float !*
- Les opérateurs numériques (+, -, *, / etc.) ne sont pas définis pour les types byte, char et short. **Java met en place une promotion numérique vers le type** int.

On retrouve les mêmes opérateurs qu'en C/C++ (même l'opérateur de cast), avec les différences suivantes :

- L'opérateur % fonctionne également pour les entiers négatifs et les flottants.
- La division par zéro pour les entiers génère une ArithmeticException ; si l'exception n'est pas interceptée, le programme s'arrête.
- Les flottants respectent les conventions IEEE 754 : il y a une représentation de l'infini positif, de l'infini négatif et d'une valeur non calculable (NaN) ; 1. / 0. donnera l'infini positif ! On peut accéder à ces représentations particulières, e.g. Float.POSITIVE_INFINITY. *Attention Float ≠ float !*
- Les opérateurs numériques (+, -, *, / etc.) ne sont pas définis pour les types byte, char et short. **Java met en place une promotion numérique vers le type** int.

On retrouve les mêmes opérateurs qu'en C/C++ (même l'opérateur de cast), avec les différences suivantes :

- L'opérateur % fonctionne également pour les entiers négatifs et les flottants.
- La division par zéro pour les entiers génère une ArithmeticException ; si l'exception n'est pas interceptée, le programme s'arrête.
- Les flottants respectent les conventions IEEE 754 : il y a une représentation de l'infini positif, de l'infini négatif et d'une valeur non calculable (NaN) ; 1. / 0. donnera l'infini positif ! On peut accéder à ces représentations particulières, e.g. Float.POSITIVE_INFINITY. *Attention Float ≠ float !*
- Les opérateurs numériques (+, -, *, / etc.) ne sont pas définis pour les types byte, char et short. **Java met en place une promotion numérique vers le type** int.

On retrouve les mêmes opérateurs qu'en C/C++ (même l'opérateur de cast), avec les différences suivantes :

- L'opérateur % fonctionne également pour les entiers négatifs et les flottants.
- La division par zéro pour les entiers génère une ArithmeticException ; si l'exception n'est pas interceptée, le programme s'arrête.
- Les flottants respectent les conventions IEEE 754 : il y a une représentation de l'infini positif, de l'infini négatif et d'une valeur non calculable (NaN) ; 1. / 0. donnera l'infini positif ! On peut accéder à ces représentations particulières, e.g. Float.POSITIVE_INFINITY. *Attention Float ≠ float !*
- Les opérateurs numériques (+, -, *, / etc.) ne sont pas définis pour les types byte, char et short. Java met en place une promotion numérique vers le type int.

On retrouve les mêmes opérateurs qu'en C/C++ (même l'opérateur de cast), avec les différences suivantes :

- L'opérateur % fonctionne également pour les entiers négatifs et les flottants.
- La division par zéro pour les entiers génère une ArithmeticException ; si l'exception n'est pas interceptée, le programme s'arrête.
- Les flottants respectent les conventions IEEE 754 : il y a une représentation de l'infini positif, de l'infini négatif et d'une valeur non calculable (NaN) ; 1. / 0. donnera l'infini positif ! On peut accéder à ces représentations particulières, e.g. Float.POSITIVE_INFINITY. *Attention Float ≠ float !*
- Les opérateurs numériques (+, -, *, / etc.) ne sont pas définis pour les types byte, char et short. **Java met en place une promotion numérique vers le type** int.

- En Java il existe un opérateur logique "ou exclusif" ^.
- Il existe une version sans court-circuit du *ou* | et du *et* &
- Alors que l'affectation = ne force pas la conversion des opérandes dans le type de la variable à gauche, l'affectation généralisée le fait.
- Il n'existe pas de surcharge des opérateurs en Java (on verra la surcharge plus tard).

- En Java il existe un opérateur logique "ou exclusif" ^.
- Il existe une version sans court-circuit du *ou* | et du *et* &
- Alors que l'affectation = ne force pas la conversion des opérandes dans le type de la variable à gauche, l'affectation généralisée le fait.
- Il n'existe pas de surcharge des opérateurs en Java (on verra la surcharge plus tard).

- En Java il existe un opérateur logique "ou exclusif" ^.
- Il existe une version sans court-circuit du *ou* | et du *et* &
- Alors que l'affectation = ne force pas la conversion des opérandes dans le type de la variable à gauche, l'affectation généralisée le fait.
- Il n'existe pas de surcharge des opérateurs en Java (on verra la surcharge plus tard).

- En Java il existe un opérateur logique "ou exclusif" ^.
- Il existe une version sans court-circuit du *ou* | et du *et* &
- Alors que l'affectation = ne force pas la conversion des opérandes dans le type de la variable à gauche, l'affectation généralisée le fait.
- Il n'existe pas de surcharge des opérateurs en Java (on verra la surcharge plus tard).

```
boolean byte char class const double final float int long
    short static void volatile

break case continue default do else for goto if return
    switch while

abstract extends implements instanceof interface new super

import native package synchronized

private protected public this transient

catch finally throw throws try
```