INTRODUCTION À

IFT287

(Thème 1)

Java - Historique

- Développé par Sun Microsystems à partir de 1991 par
  - Le canadien James Gosling
  - Et toute une équipe
- La première version est disponible à partir de 1995
- La dernière version, Java SE 8, est disponible depuis 2014
- Nous utiliserons Java SE 8 pour le cours.

Java - Principes

Java

- Doit être simple, orienté-objet et familier
- Doit être robuste et sécuritaire
- Doit être portable et indépendant de l'architecture
- Doit pouvoir exécuter du code "haute-performance"
- Doit être interprété, dynamique et *multithread*

Java - Principes

- En plus, Java
  - Est un langage sûr (fortement typé);
  - Possède une gestion automatique de la mémoire;
  - Peut-être utilisé pour faire du web.

Java - Comparaison

Comparable au C++

- Syntaxe très proche du C/C++
- Programmation orientée-objet
- Programmation structurée
- Fortement typé
- Types de base
- Gestion des tableaux
- Exceptions

Différences

- Interprété
- Gestion implicite de la mémoire
- Existence d'une classe de base Objet
- Bibliothèque de base très développée
- Pas de gestion explicite des pointeurs
- Pas de surcharge d'opérateurs

Premier exemple

- Code java

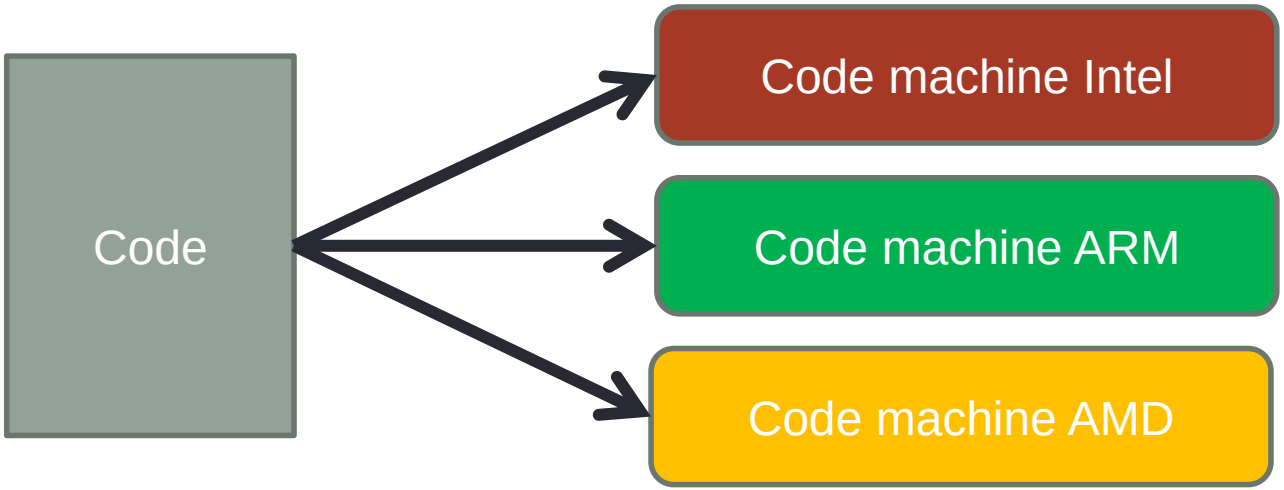
```
import java.lang.*
class PremierProg {
   public static void main (String[] args) {
    System.out.println("Bonjour!");
   }
}
```

- Code C++

```
#include <iostream.h>
int main (int argc, char *argv[]) {
   cout << "Bonjour!" << endl;
}
```

Compilation / Interprétation

Compilation d'un programme (C++, Assembleur, etc.)



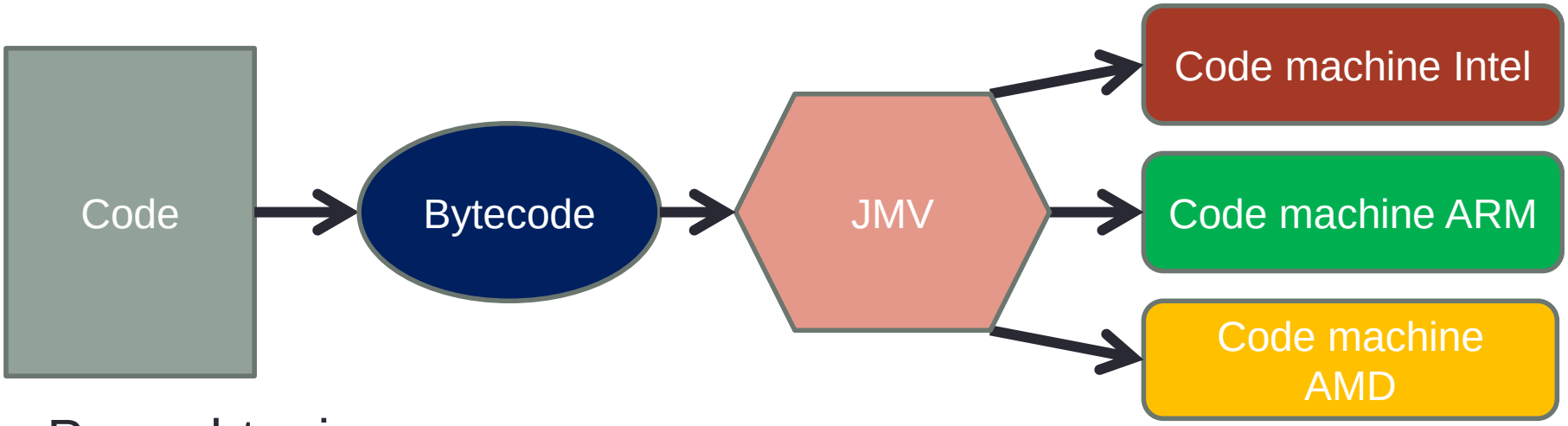
Pour obtenir un programme :

- Pour du C++, on compile directement le code en code machine

```
gcc programme.cpp
```

Compilation / Interprétation

Exécution d'un programme interprété(Java, C#, etc.)

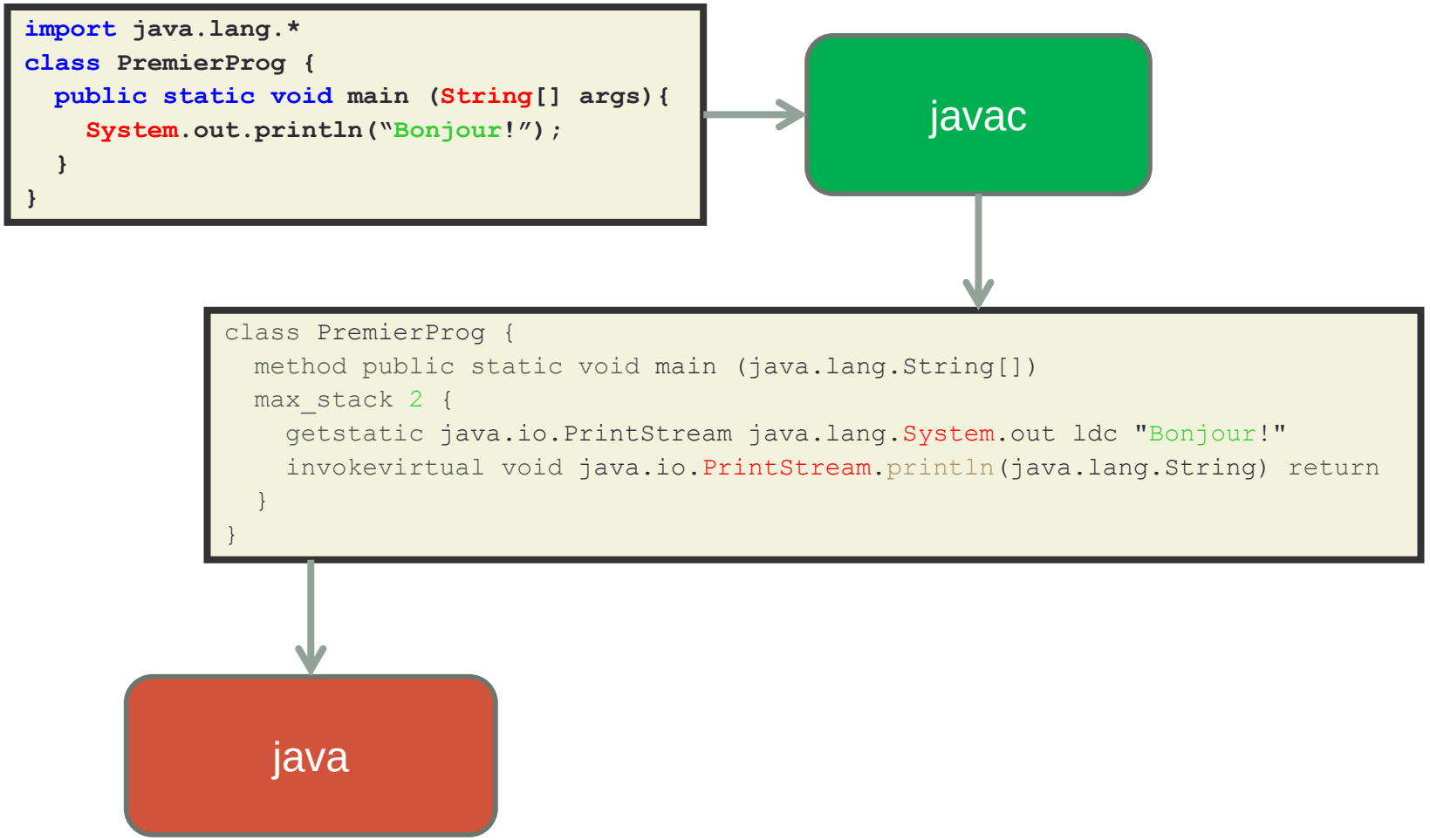


Pour obtenir un programme :

- Pour du Java, on compile le code en *bytecode*. La machine virtuelle transforme le *bytecode* en code machine au fur et à mesure que le programme est exécuté.

```
javac PremierProg.java
java PremierProg
```

Obtention d'un exécutable



Conventions

- Fichier : porte le nom de la classe qu'il décrit

```
PremierProg.java
```

- Package : commencent par une minuscule

```
java.lang.*
```

- Classe : commencent par une majuscule

```
String
```

- Variable : commence par une minuscule

```
maVariable
```

- Constante : en majuscule

```
UNE_CONSTANTE
```

Types

- Types primitifs
  - Entier
  - Caractère
  - Point flottant
  - Booléen
- Types références
  - Classes
  - Tableaux

Types primitifs

boolean : **true** ou **false**

byte : -128 à 127

short : -32768 à 32767

int : -2 147 483 648 à 2 147 483647

long : -9223372036854775808 à 9223372036854775807

char : **\u0000** à **\uffff**

float : 1e-45 à 1e+38

double : 1e-324 à 1e+308

Types références

- Pointeurs vers des objets
- Allocation avec `new`
- Pas de destructeur
  - Le ramasse-miette (*garbage collector*) de la machine virtuelle s'occupe de libérer la mémoire.

Tableaux

- Semblable au C++

```
int numero[] = new int[3];
```

- L'indexation commence à 0

- Création implicite d'une classe
  - Accès à la longueur par le champ `length`

```
numero.length;
```

- Tableaux multidimensionnels

```
int[][][] a = new int[l][m][n];
```

Structures de contrôle

- Les mêmes qu'en C++ :

  `if (CONDITION) … else …`

  `while / for / do … while`

  `switch / case`

  `goto` → À ne pas utiliser, sous peine de mort!

- On retrouve aussi :

  `break`

  `continue`

Lecture/Écriture

- Pour écrire à la console

```
System.out.println("Texte");
```

- Pour lire à la console

```
BufferedReader reader = new BufferedReader(new
                        InputStreamReader(System.in));
String ligne = reader.readLine();
```

Paradigme « orienté-objet »

- Caractéristiques :
  - Encapsulation;
  - Héritage;
  - Polymorphisme.

- Les entités sont des « ***objets*** » dont le modèle est une « ***classe*** »

- On fait agir les objets en activant des *méthodes*

Les classes

- Utilisation du mot-clé new pour créer une nouvelle instance (objet) d'une classe

```
String maChaine = new String("Bonjour");
```

- Pour les types de base (et les chaines de caractères), on peut aussi seulement assigner la valeur

```
String maChaine2 = "Bonjour";
```

```
int monEntier = 42;
```

Les classes

- Champs
- Méthodes
- Constructeurs
- Destructeurs (*`Finalizer`*)
- Visibilité
  - `public`
  - `protected`
  - `private`
  - `package`

Privilège d'accès

- Une classe peut être déclarée
  - `public`
  - `package` (sans modificateur d'accès)

- Un membre peut être déclaré
  - `public`
  - `protected`
  - `private`
  - `package` (sans modificateur d'accès)

Privilège d'accès

	Accessibilité			
	Classe	Package	Sous-classe	Monde
`public`	✓	✓	✓	✓
`protected`	✓	✓	✓	✘
`package`	✓	✓	✘	✘
`private`	✓	✘	✘	✘

* La sous-classe est dans un package différent

Les classes - Méthodes

- Les méthodes sont appelées à partir d'un objet

```
obj.methode(param1, param2);
```

- Comme en C++, une méthode peut recevoir des paramètres et retourner ou non une valeur

```
char c = maChaine.charAt(2);
```

Exemple

```
public class Exemple
{
  public static void main(String[] args)
  {
    Personne p1 = new Personne("Angelina Joli");
    Personne p2 = new Personne("Brad Pitt");
    Personne[] personnes = new Personne[2];
    personnes[0] = p1;
    personnes[1] = p2;
    p1.setConjoint(p2);
    p2.setConjoint(p1);
    …
  }
}
```

Égalité entre objets

- Égalité sur les références

```
p1 == p2
```

- Égalité sur le contenu

```
p1.equals(p2)
```

  - La méthode `equals` doit être redéfini dans chaque classe
    - Implémentation de base utilise l'opérateur ==

Passage de paramètres

- Les paramètres sont toujours passés par valeur

- Les références sont copiées
  - C'est le « pointeur » qui est copié, l'objet associé est le même
  - La méthode peut donc modifier un objet reçu en paramètre
    - Attention aux effets de bord (*side-effects*)!

- La méthode `clone` permet de copier un objet

```
obj.clone()
```

La classe `String`

- Dans le package `java.lang`
- Représente une chaine de caractères **immutable**

```
String prenom = new String("Vincent");
```

- Il est conseillé d'utiliser les valeurs Unicode pour mettre des accents (`\uXXXX`)
- Création de chaine dynamique

```
String s = "Mon nom est " + prenom + " !";
```

La classe `String`

```
boolean equals(Object o)

int compareTo(Object o)

String concat(String str)

boolean endsWith(String suffix)

boolean startsWith(String prefix)

char[] toCharArray()

String substring(int beginIndex)
```

…

La classe `StringBuffer`

- Dans le package `java.lang`

- Représente une chaine de caractères **mutable**

```
StringBuffer prenom = new StringBuffer("Vin");
        prenom = prenom.append("cent");
```

La classe StringBuffer

```
StringBuffer append(String s)

StringBuffer delete(int start, int end)

StringBuffer insert(int offset, String s)

StringBuffer reverse()

String substring(int start, int end)

String toString()
```

…

Les types génériques

- Permettent de spécifier des paramètres dénotant le type à utiliser dans une classe

- Les collections sont l'exemple classique

```
Collection<E>
Set<E>
List<E>
Map<K, V>
Vector<E>
```

La classe `Vector<E>`

- Dans le package `java.util`

- Représente un tableau dynamique

```
Vector<String> noms = new Vector<String>();
noms.add("Vincent");
noms.add("Marc");
```

La classe Vector

```
boolean add(E o)

boolean contains(Object o)

E get(int index)

E set(int index, E o)

boolean isEmpty()

boolean remove(Object o)

E lastElement()
```

…

Mot clé `final`

- Appliqué à une classe
  - La classe ne peut pas être utilisée comme parent d'héritage
- Appliqué sur une méthode
  - La méthode ne peut pas être surchargée
- Appliqué sur une variable
  - La variable ne peut être (et doit être) assignée qu'une seule fois
    - Directement à la déclaration
    - Dans le constructeur
  - Attention : Une variable `final` sur un objet référencera toujours cet objet, mais les propriétés de l'objet peuvent changer

Mot clé `static`

- Permet de ne pas créer d'instance d'une classe pour utiliser ses méthodes
- Les méthodes ou variables `static` sont communes à tous les objets de la classe
- On utilise le nom de la classe pour accéder aux membres `static` (et non une variable qui référence un objet)

```
MaClasse.maMethodeStatique()
```

La classe `java.lang.Math`

- La classe `java.lang.Math` fournit des méthodes pour effectuer des calculs mathématiques
- Les méthodes sont toutes statiques
- Permet de les appeler sans instancier la classe

```
double x = Math.acos(10);
double y = Math.sqrt(50.5);
```

Les exceptions

- Mécanisme permettant de gérer facilement les cas d'erreurs
- Même fonctionnement qu'en C++
- Toutes les exceptions héritent de la classe `java.lang.Exception`
- Attention : Les exceptions réduisent beaucoup les performances d'exécution

Les exceptions

```
public void Exemple()
{
   try
   {
      …
   }
   catch(Exception e)
   {
      System.out.println("Erreur!");
   }
   finally
   {
      …
   }
}
```

Les exceptions

```
Exception in thread "main" java.lang.NullPointerException
        at Film.addActeur(Film.java:70)
        at Exemple.main(Exemple.java:20)
```

Ligne 70

```
public void addActeur(Personne p)
{
  acteurs[nbActeur] = p;
  ++nbActeur;
}
```

Film.java

```
Film f;
f.addActeur(new Personne("Brad Pitt"));
```

Ligne 20

Exemple.java

La documentation

- Les APIs de Java sont très développées
- La documentation est disponible en ligne sur le site d'Oracle
  - https://docs.oracle.com/javase/8/docs/api/index.html
- N'ayez pas peur d'aller rechercher sur Google!

Les packages courants

- `java.lang`
  - Classes fondamentales de Java
- `java.util`
  - Classes utilitaires, collections, date et heure, etc.
- `java.math`
  - Classes pour calcul à précision arbitraire (BigInteger, etc.)
- `java.io`
  - Entrées/sorties système, flux de données, etc.
- `java.sql`
  - Accès aux sources de données

Conclusion

- Java est un langage de programmation par objets
  - Basé sur les classes
- Langage très simplifié
  - Types de base
  - Objets
  - Tableaux
- Pas de fonctions globales
- Pas de `struct`, `union`, etc.

Conclusion

- Java est un langage de programmation interprété
  - Compilation vers du *bytecode*
  - Peut quand même être compilé en code machine
- Possède un ramasse-miette pour la gestion mémoire
- Fournit directement des fonctionnalités de gestion de la concurrence (`threads`, `synchronized`, etc.)