POO-IG

Programmation Orientée Objet et Interfaces Graphiques

Cristina Sirangelo
IRIF, Université de Paris
cristina@irif.fr

Informations pratiques

- **Cours** à distance en direct (tous les **mercredis 14h-15h45**)

  https://bbb-front.math.univ-paris-diderot.fr/recherche/cri-0ik-4og-jbm

  - Seances enregistrées

- **TD** en présentiel, selon EDT

- **TP** soit en présentiel soit en mode hybride (présentiel/distantiel) par demi-groupes (precisions à venir)

Informations pratiques

- Page **Moodle** du cours :
- https://moodle.u-paris.fr/course/view.php?id=1631
  - pour :
    - Les supports de cours
    - Les sujets de TD/TP
    - La soumission du projet
    - Les annonces et les informations de dernière minute
- **S'inscrire !**
- Page "miroir" en cas de problèmes d'accès à Moodle : http://samvangool.net/pooig-2020/
- Le travail réalisé à chaque TP doit être rendu sur Moodle
  - 1 semaine de temps pour chaque TP

Contrôle des connaissances

Contrôle continu intégral

- Epreuves :
  - **CCTD :** un contrôle dans le créneau du TD10
  - **TP :** une note de TP (notation de quelques séances de TP choisies aux hasard)
  - **Projet :** en binôme, soutenance en fin de semestre
  - **CT :** un contrôle terminal, fin de semestre

- Notation
  - 1ère session : **20 % CCTD, 15% TP, 30% Projet, 35% CT**
  - 2ème session : **100% contrôle de session 2**

Contrôle des connaissances

Prise en compte des absences dans la note de session 1

- >= 2 ABI ⇒ note finale ABI
- >= 3 ABJ ⇒ note finale ABJ
- sinon (au plus 2 ABJ et au plus 1 ABI)
  - S'il y a des ABJ, le coeff le plus élevé parmi les coeff des épreuves ABJ est redistribué en parties égales sur les 3 autres épreuves
  - Note calculée sur les épreuves restantes (3 ou 4) avec tous les AB (ABI ou ABJ) replacés par 0

Plan du cours (susceptible de variation)

1 Introduction générale:
but de la programmation orientée objet, la plateforme Java
2 "Rappels":
visibilité des noms, classe, objet, instance
références et représentation mémoire: pile, tas
éléments de classe et d'instance
constructeurs, appels de méthode et passage des paramètres
surcharge, controle d'acces, encapsulation
3 Héritage
4 Interfaces et classes abstraites
5 Classes internes et expressions lambda
6 Exceptions
7 Introduction aux interfaces graphiques
8 Génericité
9 Compléments: Collections / Design Patterns / Packages et Modules

Bibliographie

De nombreux livres sur POO et java (attention aux versions de java, la version actuelle est la version 14)

- C.S.Horstman - Core Java - Prentice Hall - 11e édition
- E.Gamma, R.Helm, R.Johnson, J.Vissides - Design Patterns Elements of Reusable Object-Oriented Software

- Java en ligne:
  - API: https://docs.oracle.com/en/java/javase/14/docs/api/index.html
  - Java tutorial: http://docs.oracle.com/javase/tutorial (Attention : écrit pour Java 8)

Message

- Lien pour le cours de **Conduite de projet** de 16h

https://bbb-front.math.univ-paris-diderot.fr/recherche/ald-39c-cmx-7pe

Introduction à la programmation orientée objet

Développement de logiciel

- Problème du logiciel :
  - Taille
  - Coût : développement et maintenance
  - Fiabilité
- Solution : **Modularité**
  - Certification
  - Réutilisation de logiciel

Comment l'obtenir?

Logiciel et modularité

- Histoire
  - Fonctions et procédures (60 Fortran)
  - Typage des données (70) Pascal Algol
  - Modules: données + fonctions regroupées (80) ada
  - Programmation objet: classes, objets et héritage

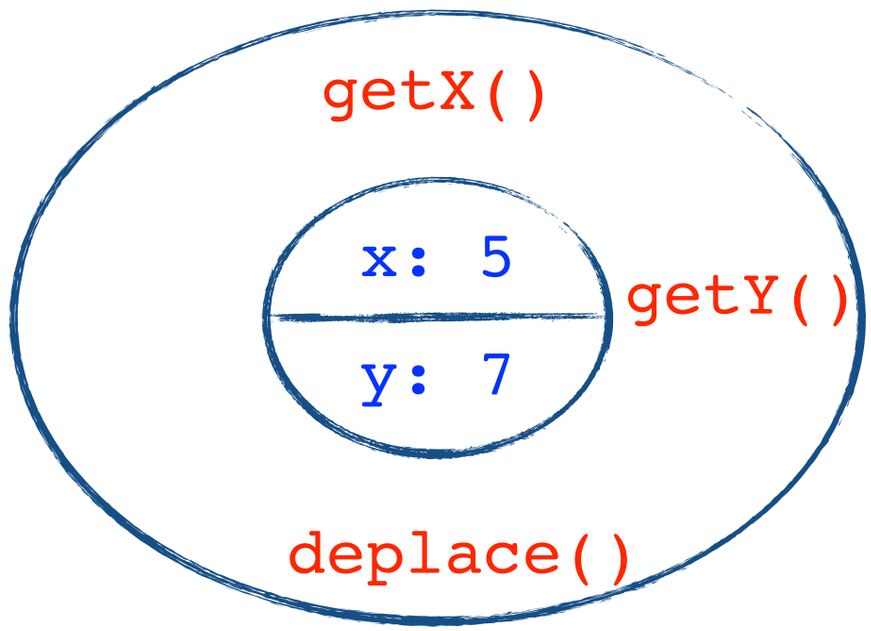
Principes de base de la POO

- Classes et Objets:
  - Classe = définitions pour des données (champs) + fonctions (méthodes) agissant sur ces données
  - Objet = élément d'une classe (instance) avec un état

Classe

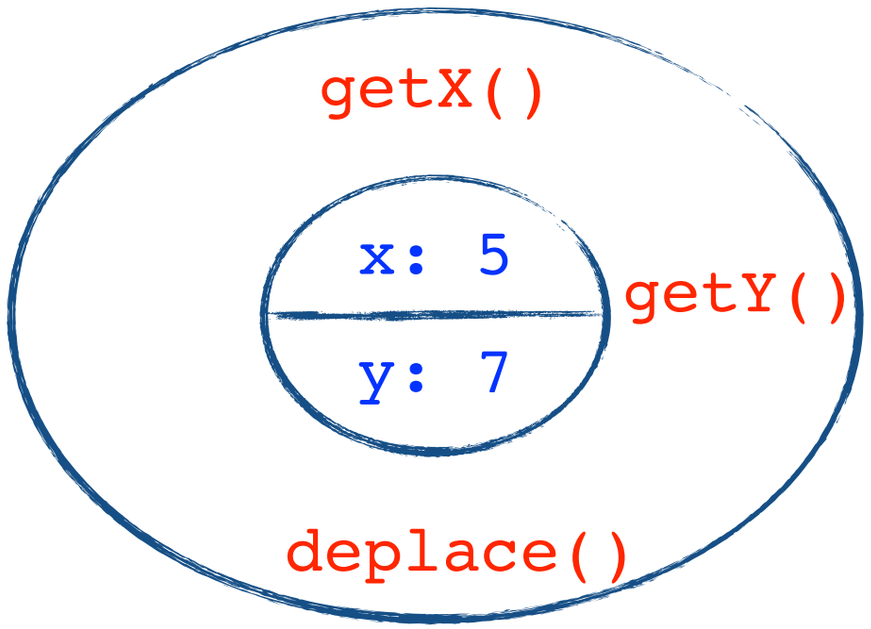
```
class Point {
  private double x, y;
  public int getX ()
  {return x;}
  public int getY ()
  {return y;}
  public void deplace
     (double newX, double newY)
  { x = newX; y = newY; }
  …
}
```

Objet de classe Point



POO et modularité

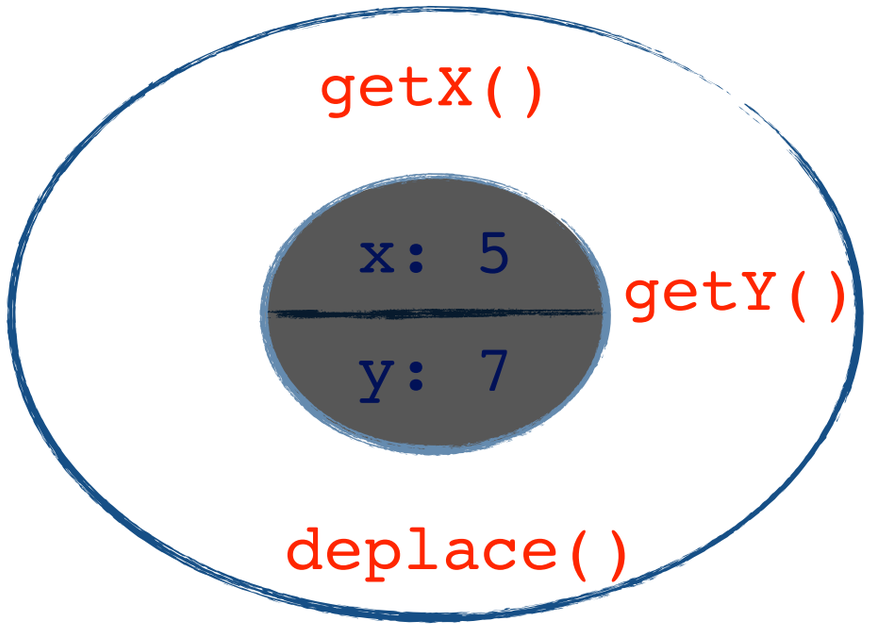
- Une classe décrit un module
  - gère des donnée, mets à disposition des méthodes pour les manipuler



- Une classe peur utiliser d'autres classes (modules) à travers ces méthodes
- => ces méthodes représentent l'"interface" avec laquelle le module peut être utilisé
- (des champs pourrait faire partie de l'interface si nécessaire)

Encapsulation

- Encapsulation :
  - en général l'état interne d'un l'objet, ainsi que ses méthodes auxiliaires, ne sont pas pas accessibles aux autres classes
  - un objet peut être manipulé uniquement par son interface



- => Séparer l'implémentation de la spécification
  - les détails d'implémentation sont « cachés » aux autres classes
  - connaître la spécification de l'interface suffit pour utiliser une classe

Séparation implémentation / spécification

- Spécification:
  - définir un type de données par son interface (exemple: empiler, dépiler pour une pile) et ses propriétés (dépiler(empiler(v)) retourne v)
  - appelé type abstrait de données (ADT en anglais)
- Implémentation
  - structure de données (exemple : tableau d'entiers, indice pour la tête de la pile, …)
  - + code qui implémente les fonctionnalités de l'interface et satisfait la spécification du type de données

Un exemple de spécification (ADT)

- Type abstrait (notation formelle):

NOM

Pile<T>

FONCTIONS

vide : Pile<T> -> Boolean

nouvelle : -> Pile<T>

empiler : T × Pile<T> -> Pile<T>

dépiler : Pile<T> -> T × Pile<T>

PRECONDITIONS

dépiler(s: Pile<T>) <=> (not vide(s))

AXIOMES

pour tout e in T, pour tout s in Pile<T>

vide(nouvelle())

not vide(empiler(e,s))

dépiler(empiler(e,s))=(e,s)

Remarques

- Les axiomes décrivent la sémantique (le comportement) des fonctions
  - Il faudrait vérifier que cette définition caractérise bien un pile au sens usuel du terme (c'est possible)

- Une spécification est une sorte de contrat entre un vendeur (la classe) et un client (le code qui utilise la classe)

Notion de contrat

- Le client ne peut utiliser l'objet que par son interface
- La réalisation de l'objet est cachée au client
- Un contrat lie vendeur et client :
  - Le contrat est conditionné par l'utilisation correcte de l'objet (pré-condition)
  - Sous réserve de la pré-condition le vendeur s'engage à ce que l'objet vérifie sa spécification (axiomes ou post-condition)

Utilisation d'une pile

- Le contrat (i.e. la spécification) suffit au client pour utiliser une pile, indépendamment de son implémentation

```
Pile<Integer> s = new Pile<Integer>();
for (int i = 0; i < 10; i++) {
        s.empiler(i);
}
for (int i = 0; i < 10; i++) {
        System.out.println(s.depiler());
}
```

- affiche

```
9
8
7
6
5
4
3
2
1
0
```

Un exemple d'implémentation d'une pile

```
import java.util.LinkedList;

public class Pile<T> {
    private LinkedList<T> items;
    public Pile(){
        items = new LinkedList<T>();
    }
    public boolean estVide(){
        return items.isEmpty();
    }
    public T empiler(T item){
        items.addFirst(item);
        return item;
    }
    public T depiler(){
        return items.removeFirst();
    }
}
```

- Implémentations alternatives : par tableau + indice top, par ArrayList, …
- Java permet également de définir plusieurs implementations de la même interface (voir plus loin)

POO et modularité : héritage

- Heritage : une autre caractéristique de la POO qui permet modularité et réutilisation du code
- Une classe peut peut être une extension d'une autre classe

```
classe Personne {
  …
}
class Employé extends Personne{
  …
}
```

- Employé hérite toutes les propriétés et méthodes de Personne (nom, prénom, nss, adresse, …) - réutilisation
- elle représente et manipule uniquement la "différence" (salaire, département, employeur,…)

- Polymorphisme : un Employé est une Personne => un objet Employé peut être utilisé partout où on s'attend une Personne (mais pas vice-versa)

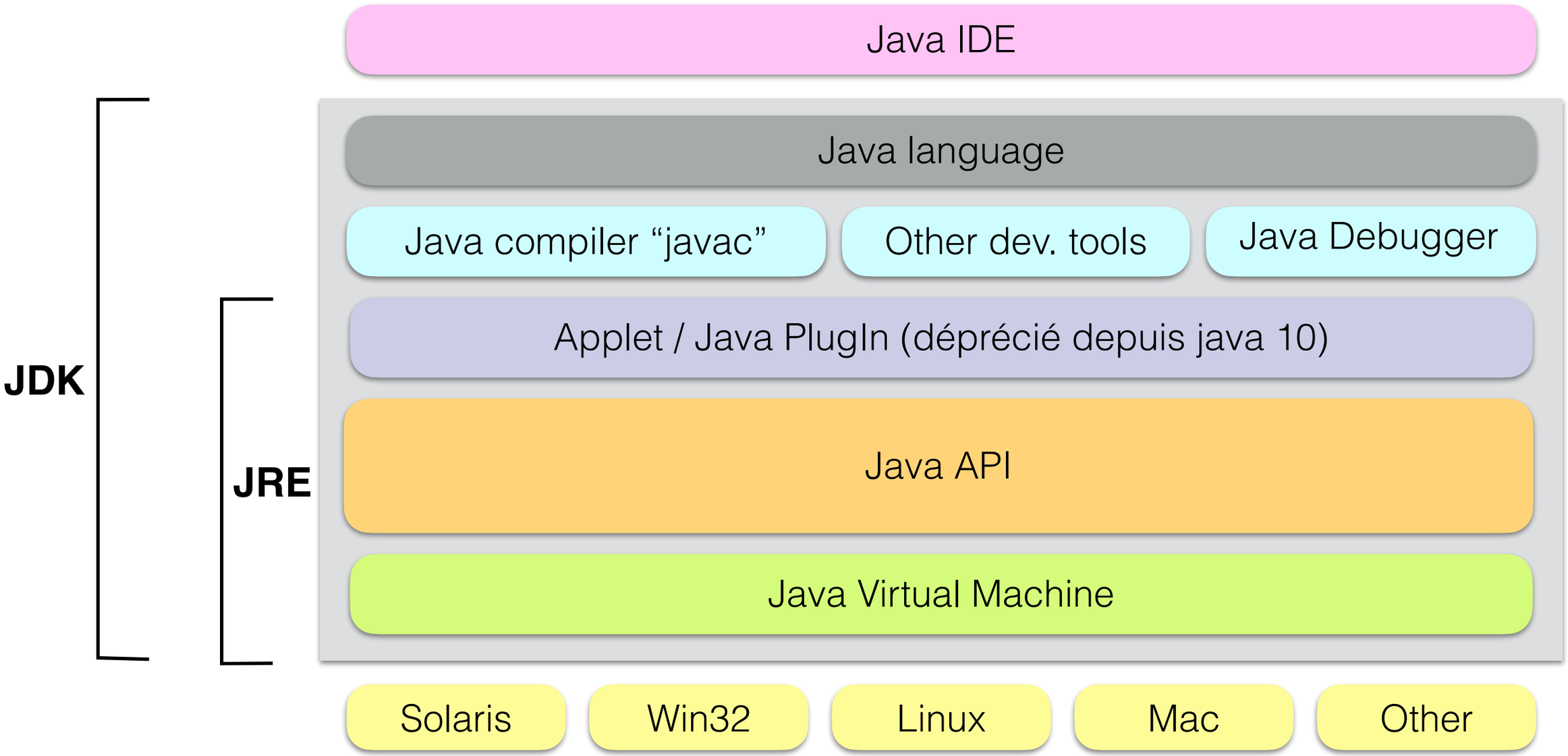
Introduction à Java

Java

- Plus qu'un langage de programmation, une vrai plateforme:
  - Un langage
  - Une bibliothèque très vaste (Java API*)
  - Un environnement d'exécution portable (Java VM + Java Plugins)

* API : Application Programming Interface

Tout un environnement…



Tout un environnement…

- JDK (anciennement SDK) : Java Développement Kit
  - logiciel qui permet le développement et executions de programmes Java
  - telechargeable à http://www.oracle.com/technetwork/java/javase/downloads
- JRE : Java Runtime Environment
  - logiciel qui permet uniquement l'exécution de programmes java
  - contient java API et les outils de déploiement et exécution (JVM)

- IDE : Integrated Development Environment
  - Logiciel de support au développement. Alternative haut-niveau au développement par éditeur de texte + outils en ligne de commande (javac + java)
  - Pas nécessaire, mais utile pour des gros projets
  - Certains gratuits : NetBeans, Eclipse, …

Les éditions de la plateforme Java

- Standard edition (SE)
  - pour le développement d'applications standard
- Enterprise edition (EE)
  - pour le développement d'applications serveurs
- Micro Edition (ME)
  - pour le développement de logiciel embarqué

- Dans ce cours : Standard Edition (Java SE)

Evolution de la plateforme Java

Nom de la plateforme	Version JDK	année	Nouveautés dans le langage	# de classes/ interfaces
Java 1.0	jdk 1.0	1996	le langage	211
Java 1.1	jdk 1.1	1997	les classes `inner`	477
Java 2	jdk 1.2	1998	Le modificateur `strictfpr`	1524
	jdk 1.3	2000	Aucune	1840
	jdk 1.4	2002	Assertions	2723
Java 5.0	jdk 1.5	2004	classes generiques, boucles “for each”, `varargs`, autoboxing, metadata, enumerations, static import	3279
Java 6	jdk 1.6.0	2006	Aucune	3793
Java 7	jdk 1.7.0	2011	`Switch` avec chaines, opérateur “diamond”, littéraux binaires, meilleure gestion des exceptions	4024
Java 8	jdk 1.8.0	2014	expressions Lambda, interfaces avec methodes par defaut, bibliothèques stream et date/time	4240
Java 9	jdk 9.0.0	Sept 2017	module system, @SafeVargs on private methods, effectively final var in try-with-resources, diamond with anonymous classes, private interface methods, deprecated Java Applet API et Java Plug-in	6005
Java 10	jdk 10.0.0	Mars 2018	pas de nouveautés majeures	6002

Evolution de la plateforme Java

Nom de la plateforme	Version JDK	année	Nouveautés dans le langage	# de classes/ interfaces
Java 11	jdk 11.0.0	Sept 2018	Dynamic class-file constants; Local parameter type inference for lambdas; new String methods; new Stream methods; new toArray methods for collections; JavaFX, Java EE and CORBA modules removed from JDK; new garbage collector	
Java 12	jdk 12.0.0	Mars 2019	improved garbage collection; no major language features	
Java 13	jdk 13.0.0	Sept.2019	Memory optimization, new switch expressions, text blocks	
Java 14	jdk 14.0.0	Mars 2020	Memory optimization, new switch expressions, text blocks, pattern matching for instanceOf, Detailed NullPointerException, Records	

Caractéristiques principales de Java

- Indépendant de la plateforme
- Langage interprété et byte code
- Syntaxe à la C
- Orienté objet (classes héritage)
- Pas de pointeurs! (ou que des pointeurs!)
    - Ramasse-miettes (garbage collector)

- Multi-thread
- Distribué (WEB et serveurs) …
- Dernière version Java SE 14
    - Site:http://www.java.com/fr

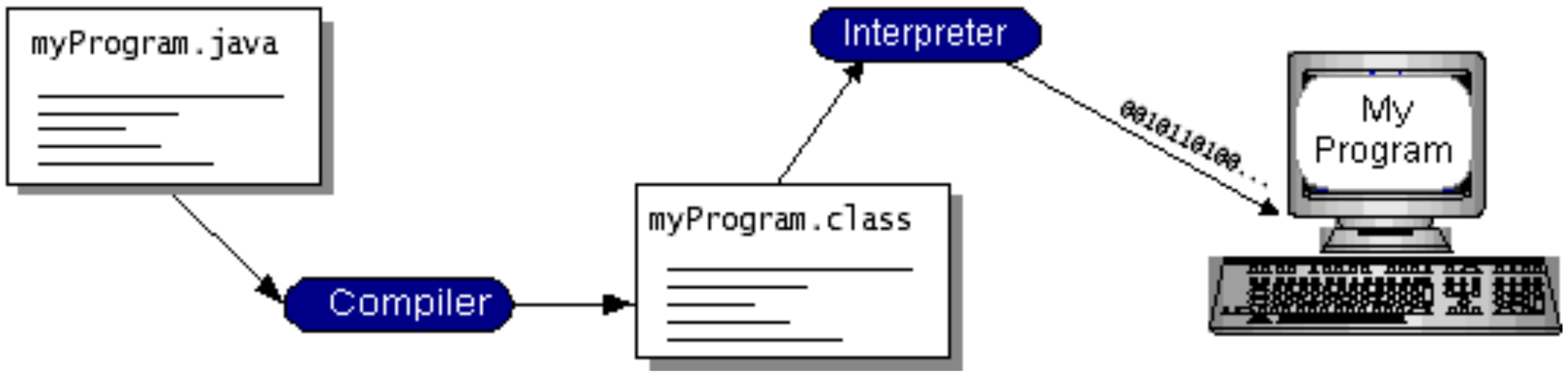
- À l'origine gratuit, depuis Avril 19 payant pour utilisation commerciale. Mais version OpenJDK disponible sous licence GPL

Language interpreté

- La compilation génère un fichier .class en « bytecode » (langage intermédiaire indépendant de la plateforme).
- Le bytecode est interprété par un interpréteur Java JVM

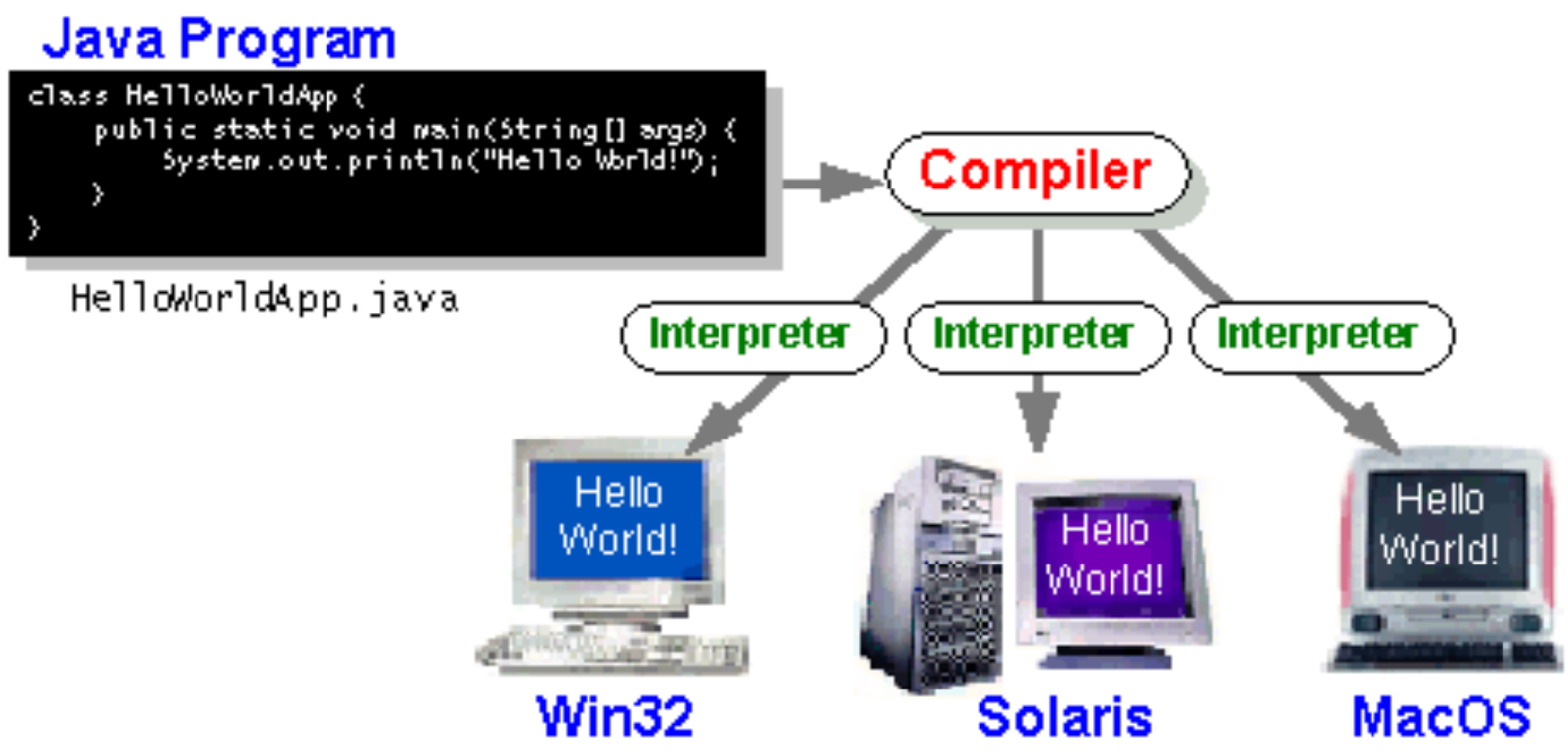
Compilation commande `javac`

Interprétation commande `java`



Langage intermédiaire et Interpréteur…

- Avantage:
  indépendance de la plateforme ("Write once, run everywhere")
  - en particulier : échange de byte-code



- Inconvénient: efficacité

Interpréteur et efficacité

- Interpréteur en principe :
  - traduit chaque instruction du byte-code à la volé en code machine (dépendant de la plateforme hardware) pendent l'exécution
    - Différent de la compilation (traduit tout, puis exécute le code généré)

- Interpréteur en pratique :
  - pour des questions d'efficacité, les fragments de code le plus utilisés sont traduits entièrement avant l'exécution et optimisés (compilateurs just-in-time)

Créer une application

- Fichier Appli.java:

```
/* Une application  basique...*/
class Appli {
    public static void main(String[] args) {
     System.out.println("Bienvenue en L2...");
                //affichage
    }
}
```

- Compilation: `javac Appli.java`
- Crée `Appli.class` (bytecode)
- Interpréter le bytecode: `java Appli`
- Si: Exception in thread "main" java.lang.NoClassDefFoundError:
  - Il ne trouve pas le main -> vérifier le nom!
  - Variable CLASSPATH ou option -classpath

statique-dynamique

- Compilation (=statique)
  - du code source vers un code objet
    - erreurs de syntaxe
    - résolution du typage (chaque variable-expression a un type déterminé à la compilation)
- Exécution (=dynamique)
  - exécution du code objet
    - erreur à l'exécution
    - association d'un nom de fonction à son code (voir plus loin)