DAUPHINE UNIVERSITÉ PARIS | MIDO MATHÉMATIQUES ET INFORMATIQUE DE LA DÉCISION ET DES ORGANISATIONS

PSL RESEARCH UNIVERSITY

Licence L3 M I 2 E

JAVA

Michel Masson

V1.2

EQUIS ACCREDITED

La spécificité de Java

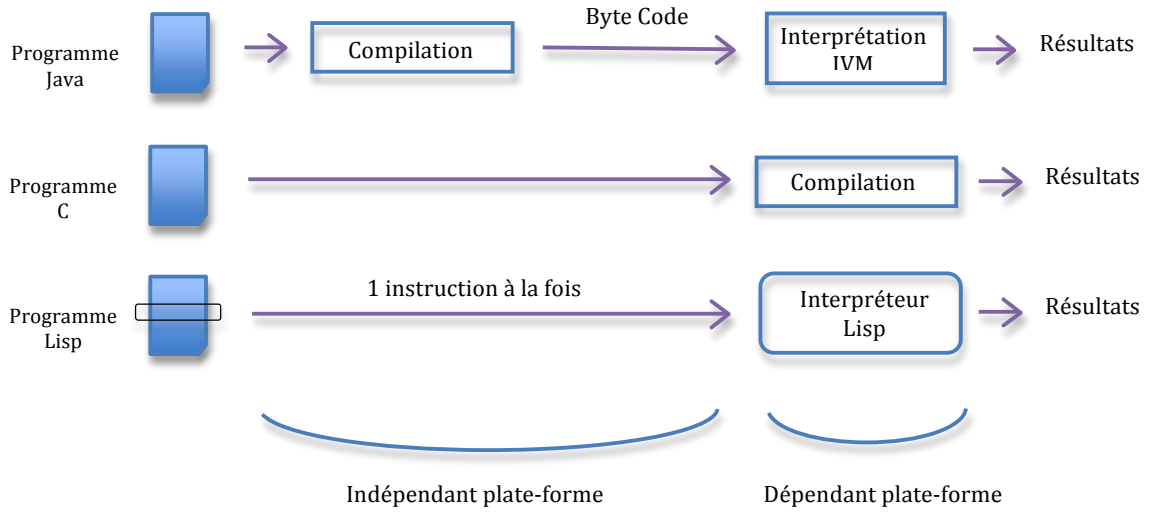
La programmation est au cœur de l'activité informatique; elle consiste à écrire une suite d'instructions qui seront exécutées séquentiellement par la machine. Initialement ces instructions étaient de bas niveau: elles variaient selon la machine utilisée. L'apparition de langages dépassant le niveau d'abstraction du langage spécifique à la machine a apporté un progrès considérable; l'écriture d'un programme devenait plus simple, plus rapide et plus facile à corriger. Ainsi sont apparus successivement Fortran, Cobol, PL/1, Basic, Pascal, C.
Apparue plus tard, la programmation orientée objet facilite la modélisation des problèmes abordés: au lieu de décomposer un programme en actions elles-mêmes décomposées en sous-actions, la programmation orientée objet décompose le problème en objets et en relations entre ces objets. La programmation objet repose sur quatre paradigmes:

- Tout est objet: un objet est une entité caractérisée par des propriétés (attributs) et des traitements (méthodes).
- Tout objet est instance d'une classe: une classe décrit un ensemble d'objets ayant des propriétés et des traitements identiques.
- Les classes sont organisées hiérarchiquement: cela permet à une classe d'hériter des propriétés des classes hiérarchiquement supérieures (cf. Cours Java Objet).
- Les objets communiquent par envoi de message.

Bien que Java (développé par SUN à partir de 1994) ne soit pas le premier langage à implémenter ces quatre paradigmes son succès s'explique par plusieurs raisons: c'est la première plateforme gratuite, maintenue, fiable et sécurisée; à cela s'ajoute la portabilité des applications. Un autre point décisif réside dans l'introduction des **servlets** ouvrant la voie à la programmation au sein des serveurs et constituant une amélioration par rapport à la programmation CGI (Common Gateway Interface).

Java: un langage semi-compilé

La façon dont est analysé et exécuté un programme induit une partition des langages de programmation en trois catégories: les langages compilés, les langages semi-compilés et les langes interprétés.
La plupart des langages de programmation sont des langages compilés: le compilateur (un programme spécifique à chaque langage et à chaque plate-forme de développement) produit un fichier exécutable qui une fois chargé en mémoire fournira les résultats attendus.
Dans le cas de Java, le compilateur transforme le programme initial en un "byte code" (c'est un programme transformé dans un langage moins évolué que Java mais indépendant de la plate-forme d'accueil), un interpréteur appelé JVM (pour Java Virtual Machine) et dépendant cette fois de la plate-forme d'accueil) exécute ce byte code pour fournir les résultats.
Les langages interprétés (Basic, Lisp) voient leurs instructions analysés et exécutées une à une pour fournir pas à pas le résultat du programme. Une instruction située dans un boucle sera donc analysée et exécutée autant de fois que la boucle compte de pas (dans un langage compilé, l'analyse a lieu une seule fois). Il s'ensuit un temps d'exécution plus long.
Le schéma suivant synthétise ces trois approches:



Il peut sembler complexe d'avoir une double conversion pour exécuter un programme Java, mais un interpréteur est beaucoup plus simple qu'un compilateur; la partie dépendante de la plate-forme de développement étant facilement développable, son usage est largement répandu. L'échange de programmes se fait donc avec une forme intermédiaire, le byte code, ce qui renforce la sécurité puisque ce n'est pas le programme échangé qui est exécuté mais son byte code via la JVM.

Un premier programme:

Convention: les mots-clés du langage et les programmes utilisent cette `police`. D'autre part, si un identificateur est formé deplusieurs mots, le début de chaque mot est signalé par une majuscule: `noClient`, `adresseEnvoiColis`, `perimetre`, désignent des identificateurs de variables, d'attribut ou de méthode.
Java est un langage orienté objet, il utilise donc des classes pour construire ces objets; tout programme Java doit donc contenir au moins une classe. Une classe contient des déclarations et des méthodes. Une des classes doit comporter la méthode `main`, c'est cette méthode qui sera lancée au démarrage.

```
class Exercice1 {
     public static void main(String argv[]) {
         // commentaire
          int a;
          a = 3;
          String s = "à tous !";
          /* commentaire
             sur plusieurs lignes */
          System.out.println ( " Bonjour " + s + '\n' + a) ;
          }
     }
```

On notera l'indentation (générée automatiquement par la plate-forme de développement), `'\n'` désigne le caractère de saut de ligne.
La sortie de ce premier programme Java est :

```
Bonjour à tous !
3
```

Remarques:

- `Exercice1` est le nom de la classe, elle est déclarée par le mot `class`
- `public` indique que la méthode `main` est visible de l'extérieur de la classe

- `static` signifie que main n'est pas associée a une instance mais à une classe
- `main` est le nom de la méthode de classe (avec un argument de type `String`)
- Une variable `a` de type entier est déclarée puis initialisée avec la valeur 3.
- Une variable `s` de type `String` est initialisée avec la chaine de caractères `"à tous !".`
- `System` est le nom de la classe système
- `println` est une méthode imprimant une ligne suivie d'un retour chariot.

Le code précédent est sauvegardé dans un fichier `Exercice1.java` (le nom du fichier doit être celui de la classe avec l'extension `java`).

Exécution d'un programme Java:

Exécution en mode commande (Windows):

Sun propose en ligne un kit de développement (JDK pour Java Development Kit) permettant la compilation et l'exécution d'un programme java. Ce kit une fois installé, il convient d'initialiser la variable PATH en ouvrant le panneau de configuration, puis en cliquant sur Système, onglet "Avancé". Cliquer sur le bouton variable d'environnement; sélectionner la variable PATH dans le panneau inférieur et cliquer sur "modifier". Il faut ajouter le chemin complet du répertoire bin du JDK (terminer par ;).

Ainsi `si C:\Sun\SDK\jdk\bin;` est le chemin du répertoire `bin`, on a:

ancienne valeur de PATH `C:\temp;`

nouvelle valeur de PATH `C:\temp; C:\Sun\SDK\jdk\bin;`

Cette manipulation permet l'accès au répertoire du kit depuis le répertoire ou sont développées les applications écrites en Java. La compilation et l'exécution se font grâce aux commandes:

```
>E:\ApplisJava javac Exercice1
>E:\ApplisJava java Exercice1
```

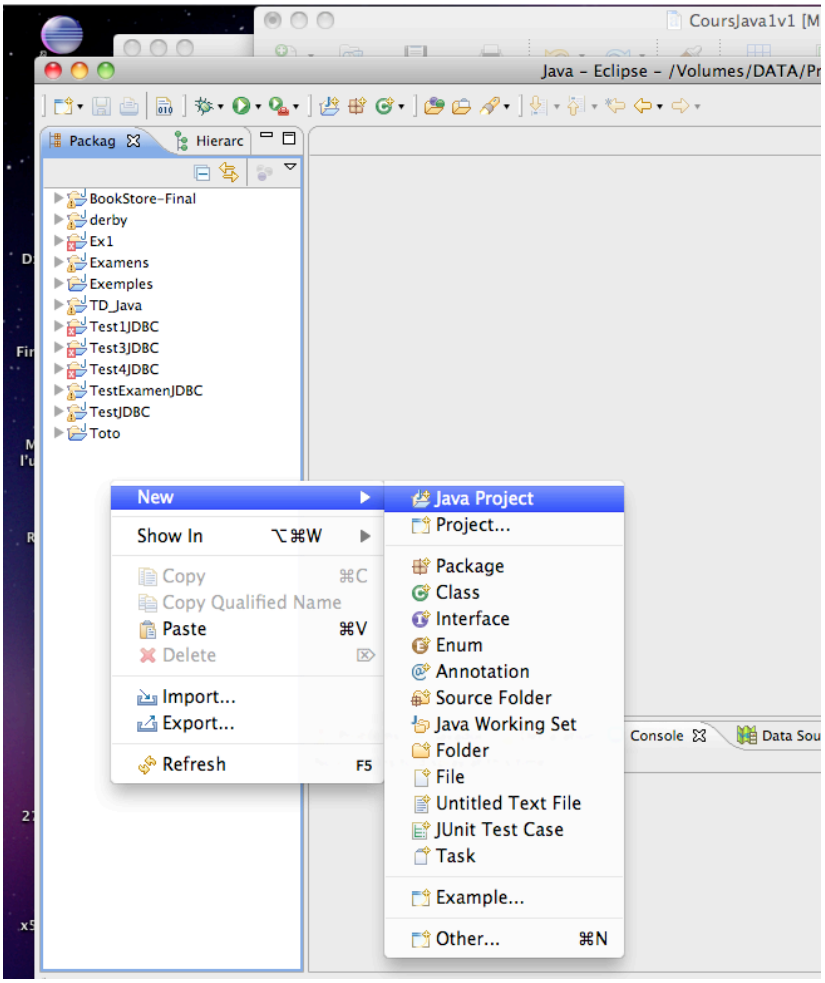
(la première commande lance la compilation et génère un byte code dans le répertoire ApplisJava, la seconde lance la JVM sur le byte code précédemment généré)

Utilisation de la plate-forme Eclipse:

Cette plate-forme fournit un cadre de travail plus agréable, les figures suivantes en illustrent le fonctionnement (www. eclipse.org).

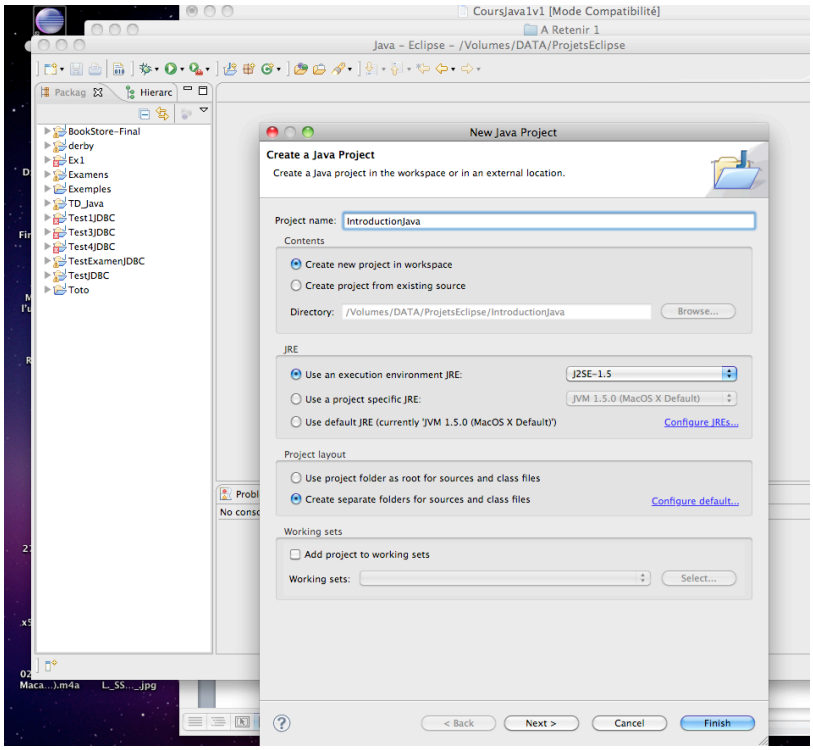
On lance l'environnement Eclipse en double-cliquant sur son icône. L'élément de base est le projet, l'ensemble de ses éléments peut être sauvegardé en vue d'un portage sur une autre plate-forme (quelque soit la plate-forme d'origine). Pour créer un projet, un clic droit dans le panneau de gauche fait apparaître un menu contextuel dans lequel on choisira "New / Java Project".

Fig. 1



La validation du choix fait apparaître la fenêtre de création de projet:

Fig. 2



Une fois le projet crée, il apparaît dans le panneau de gauche parmi les projets existants. Un clic droit sur ce projet fait apparaître le menu contextuel semblable à celui de la Fig.1 dont lequel on choisira "New / Class". On pourra cocher la création automatique de la méthode `main`. La classe avec sa méthode apparait alors dans le panneau de droite (Fig.5).

Fig. 3

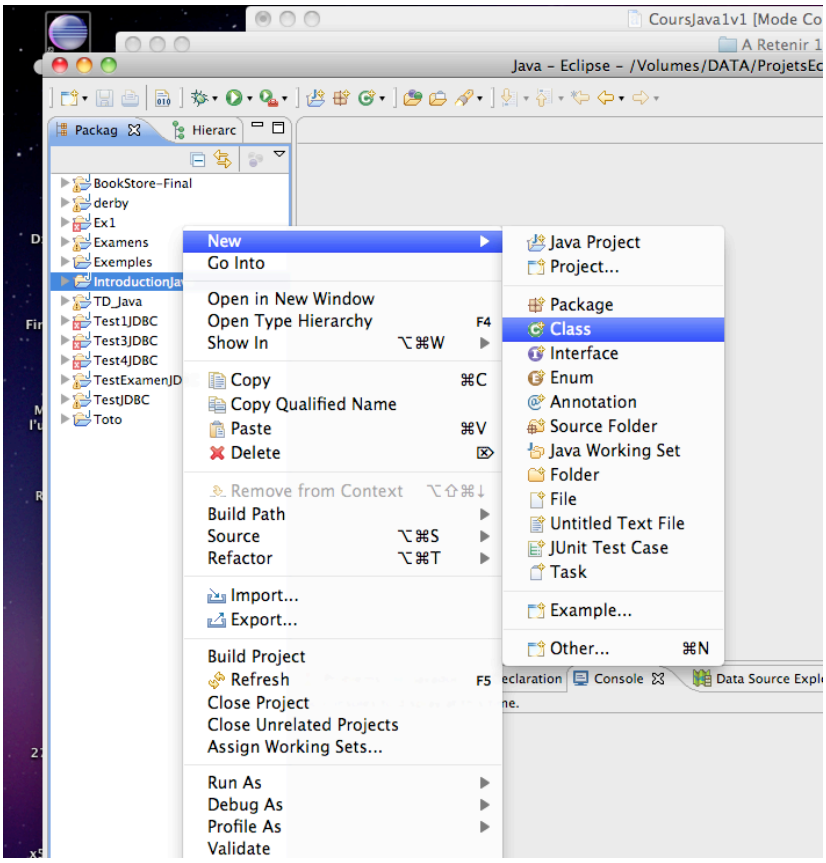


Fig. 4

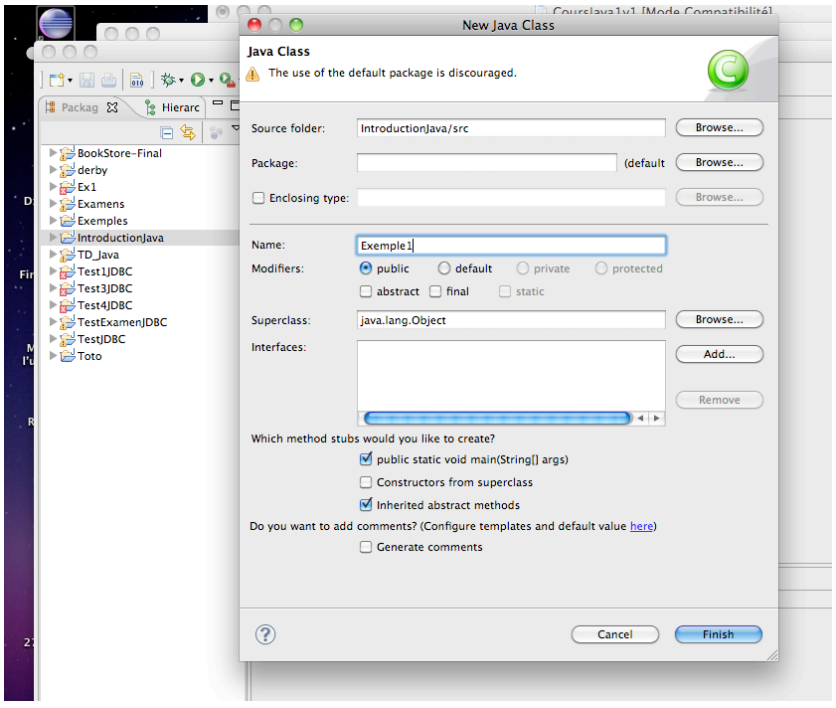
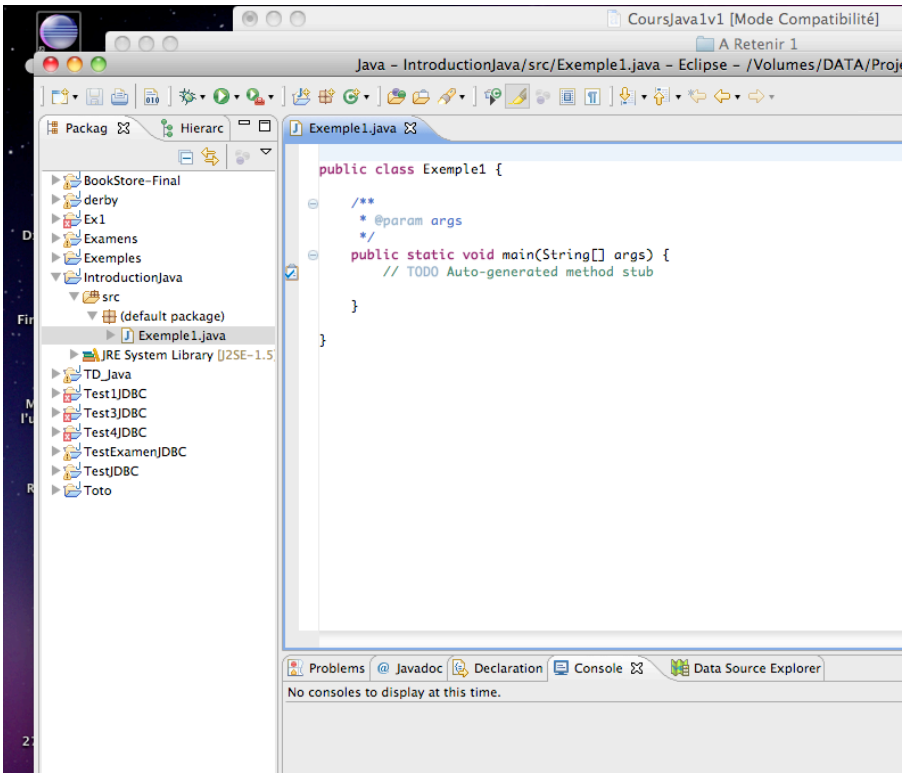
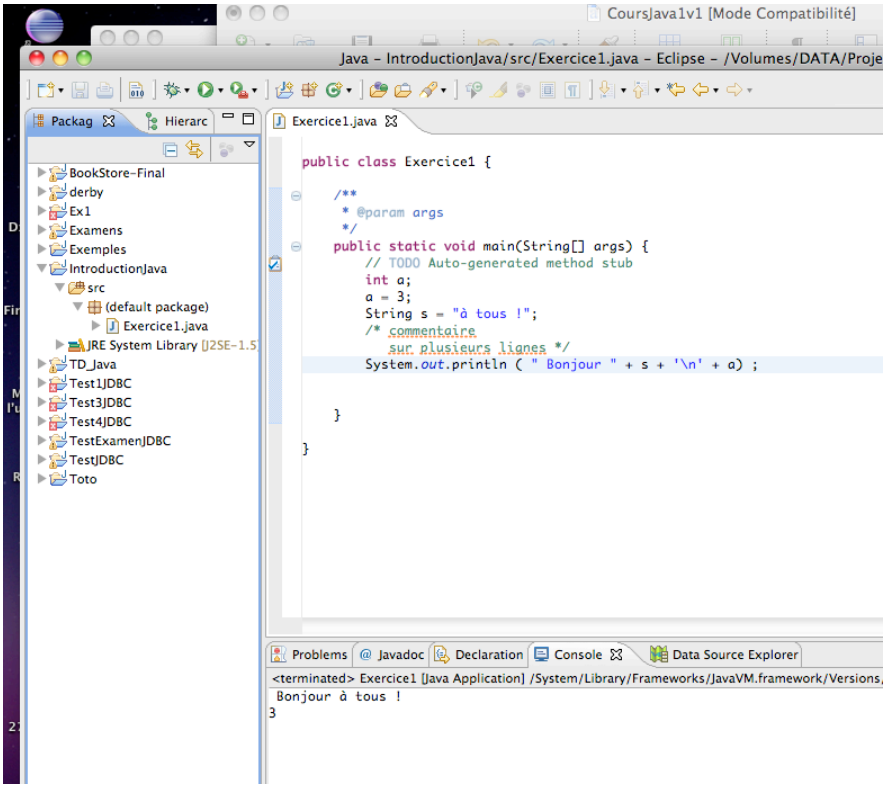


Fig. 5



Après avoir complété la méthode `main`, on peut lancer l'exécution en cliquant sur la flèche blanche sur fond vert en haut de la fenêtre ou en effectuant un clic droit sur la classe Exemple1 dans le panneau de gauche et en choisissant "Run as / Java Application". Le résultat est affiché dans le panneau inférieur (Fig. 6).

Fig. 6



Eléments du langage: Identificateurs, Variables, Constantes, types primitifs

Un identificateur sert à désigner tout élément du langage, il commence par une lettre et les mots clés ne peuvent être utilisés. La portée d'un identificateur (i.e. l'espace où il est utilisable) va de l'endroit où il est crée à la fin du bloc le contenant (le début d'un bloc est défini par "{" et sa fin par "}").

Une variable est une zone mémoire pouvant contenir une valeur typée, son identificateur permet de la manipuler dans le programme. Le type peut être une classe ou un type primitif. Il y a 8 types primitifs:

	Taille	**Champ**
`boolean`	1 bit	true, false
`char`	2 octets	0 → 65535
`byte`	1 octet	-128 → +127
`short`	2 octets	-32768 → +32767
`int`	4 octets	-2147483648 → -2147483647
`long`	8 octets	-2^{63} → $+2^{63}$ - 1
`float`	4 octets	-1.4 E-45→ +3.4 E38
`double`	8 octets	-4.9 E-324 → 1.7 E308

(aucun calcul ne peut être effectué avec les types `byte`, `short` et `char`, ils ne peuvent pas recevoir le résultat de calculs mais servent à stocker des données)

```
short s1 = 3;
short s2 = 4;
short t = s2+s3;          // incorrect
int   r = s2+s3;          // correct
short v = (short)(s1+s2); // correct
```

Les types numériques (i.e. autres que `char` et `boolean`) sont rangés dans le tableau précédent du moins général au plus général; on peut ranger une valeur d'un type donné dans une variable de type plus général. Dans le cas inverse, il est nécessaire d'utiliser l'opérateur de `cast`. Les instructions suivantes illustrent ces règles:

```
byte i =3;
int j ;
long k;
j=3;
k=j;
float l:
l = (float) j;
final int m = 5;
byte i =3;
//i = i + 1;  instruction illicite
i++;
char c = 'a';
//c = c +1;   instruction illicite
c++;
short s = 5;
//s = s + 1;  instruction illicite
s++;
int b = 9;
s = (short)b;
int j ;
long k;
j=3 + s;
k=j;
float l;
```

```
l = (float) j;
final int m = 5;
float a = 5.7f;  // saisie d'une valeur fract.
double d = 7d;   // saisie d'une valeur double précision
System.out.println("i="+i+'\n'+"c="+c+'\n'+"s="+s+'\n'+"l="+l+'\n'+"a= "+a);
```

Affichage obtenu:

```
i= 4
c= b
s= 9
l= 12.0
a= 5.7
```

Dans l'exemple précédent la variable `m` est déclarée `final`: toute variable ainsi déclarée ne peut être modifiée une fois affectée (elle peut donc être assimilée à une constante pendant toute l'exécution du programme).
A ces éléments s'ajoutent les objets: ce sont des instances de classes prédéfinies ou définies par l'utilisateur. La classe `String` est importante; elle permet la manipulation des chaînes de caractères, son utilisation est détaillée au paragraphe suivant.

La classe String et les Chaînes de caractères

La classe `String` permet de créer et de manipuler des chaines de caractères. Il y a plusieurs façons de créer ces objets (ce sont des instances de la classe, cf. Cours Java Objet):

```
String s1;
s1="Ceci est ";
String s2="un ";
String s3=new String();
s3="exemple de ";
String s4=new String("chaine");
System.out.println(s1+s2+s3+s4);
```

Résultat:

```
Ceci est un exemple de chaine
```

On remarquera que l'opérateur + désigne l'addition et la concaténation. Dans l'évaluation d'une expression, si les opérandes du premier + rencontré sont des chaînes, la concaténation restera l'opérateur par défaut (sauf parenthésage explicite) comme le montre les exécutions suivantes:

```
System.out.println("Résultat: " + 2 + 3);
System.out.println("Résultat: " + (2 + 3));
System.out.println(2 + 3 + " Résultat");
```

On obtient:

```
Résultat: 23
Résultat: 5
5 Résultat
```

Eléments du langage: les opérateurs

A partir des éléments précédents on peut construire des expressions complexes en utilisant les opérateurs suivants. Ils sont rangés par priorité décroissante, ces priorités permettent d'éviter l'usage systématique des parenthèses dans l'écriture d'une expression.

```
boolean x = ((3 *  i) >= 0) & (i<j);
```

peut ainsi être remplacé par

```
boolean x = 3 *  i >= 0 & i<j;
```

++ , -- postfixés	opérateurs de postincrémentation

++ , -- préfixés	opérateurs de préincrémentation
()	cast
* / %	
+ -	
<<>>	opérateurs de décalage
< <= > >=	opérateurs de comparaison
==!=	test sur l'égalité et la différence
&	ET logique
|	OU logique
&&	ET logique optimisé
||	OU logique optimisé
? :	opérateur conditionnel
=, +=, -=, *=, /=, <<=, >>=	

Remarques:

`a<<2` cette opération décale de 2 bits à gauche le contenu de `a` (des 0 apparaissent à droite).

L'opérateur conditionnel permet l'affectation conditionnelle d'une variable:

`max = a>b ?a:b;` // affecte à la var. max le plus grand des élts `a` et `b`

`abs = x>0 ?x:-x ;` // affecte à la variable abs la valeur absolue de `x`

Les expressions `i=i+1` et `i=i-1` sont courantes en programmation; on dispose en Java comme en C d'opérateurs d'incrémentation et de décrémentation. En Java, on peut préciser le moment de cette incrémentation / décrémentation: dans `++i` la valeur de `i` est d'abord incrémentée avant son utilisation, dans `i++` la valeur de `i` est utilisée puis incrémentée comme le montre l'exemple suivant:

```
int i1 =5;
int i2 =5;
int j = ++i1+10;
int k =i2++ + 10;
System.out.println("résultat avec préincrémentation j= "+j+" i1= "+ i1);
System.out.println("résultat avec postincrémentation j= "+k+" i2= "+ i2);
```

L'affichage donne:

```
résultat avec préincrémentation j= 16 i1= 6
résultat avec postincrémentation j= 15 i2= 6
```

Les. opérateurs de postincrémentation sont appliqués après l'opération concernée par leur opérande. Les opérateurs de prédécrémentation et de postdécrémentation obéissent aux mêmes règles.

Les opérateurs logiques optimisés n'évaluent pas la seconde opérande lorsque la première suffit à prédire le résultat: si a vaut 7 au moment de l'évaluation de (a < 5) && (b < 10), l'expression (b < 10) n'est pas évaluée.

Les opérateurs +=, -=, *=, /= sont des simplifications d'écriture correspondant à:

`variable op= expression` à la place de `variable = variable op expression`

ex:

`a += 3` pour `a = a + 3`

Eléments du langage: les instructions

Dans ce qui suit, `action action1 action2` désignent soit une instruction simple soit un ensemble d'instructions, dans ce cas l'ensemble des instructions constitue un bloc délimité par { et }.

IF

```
if (condition) action1; [ else action2; ]
```
La partie entre crochets est optionnelle.
```
if (i <2) i++; else {j = i;k = 0;}
if (i <2) i++; else j = i;k = 0;
```
dans la deuxième ligne, l'affectation de k est réalisée dans tous les cas.

SWITCH
```
switch(expression)
case valeur1: action1;
case valeur2: action2;
default: action;  // action exécutée si expression ne prend
                  // aucune des valeurs envisagées par case.
```

WHILE
```
while (condition) action;
```

Remarque: on prendra garde aux effets de bords induits par l'usage d'un opérateur ++ ou – dans la condition.
```
int a = 3;
while(--a>0)System.out.println("a= " + a);
int b = 3;
while(b-->0)System.out.println("a= " + b);
```

Résultats:

```
a= 2
a= 1
a= 2
a= 1
a= 0
```

DO
```
do action; while (condition);
```

Boucle For
```
for (initialisations:condition:incrémentation) action;
```

Exemple:
```
for(int i=0;i<3;i++)System.out.println("i= "+i);
```
affichera:
```
i= 0
i= 1
i= 2
```

Eléments du langage: les tableaux

Les tableaux sont des objets du langage, ils doivent donc être déclarés avec l'opérateur new (comme les chaines de caractères). Les éléments d'un tableau peuvent être des types primitifs, des objets ou même des tableaux (cas d'une matrice). On accède aux éléments d'un tableau par le nom du tableau et par un ou des indices, les indices commencent à 0.

```
int[] tab;
tab = new int[4];
tab[0]=3; tab[1]=4;
int suite[] = {1,2,3,4,5,6,7,8,9};
int mat[][] = {{1,2},{3,4},{5,6}};
String semaine[]={"Lundi","Mardi","Mercredi","Jeudi","Vendredi","Samedi","Dimanche"};
System.out.println("tab[1]= "+tab[1]+", suite[1]= "+suite[1]+
            ", mat[1][1]= "+mat[2][1]+" ,semaine[6]= "+semaine[6]);
System.out.println(tab.length+" "+suite.length+" "+mat.length+" "+mat[1].length);
```

On obtient:
```
tab[1]= 4, suite[1]= 2, mat[2][1]= 6 ,semaine[6]= Dimanche
```

```
4 9 3 2
```

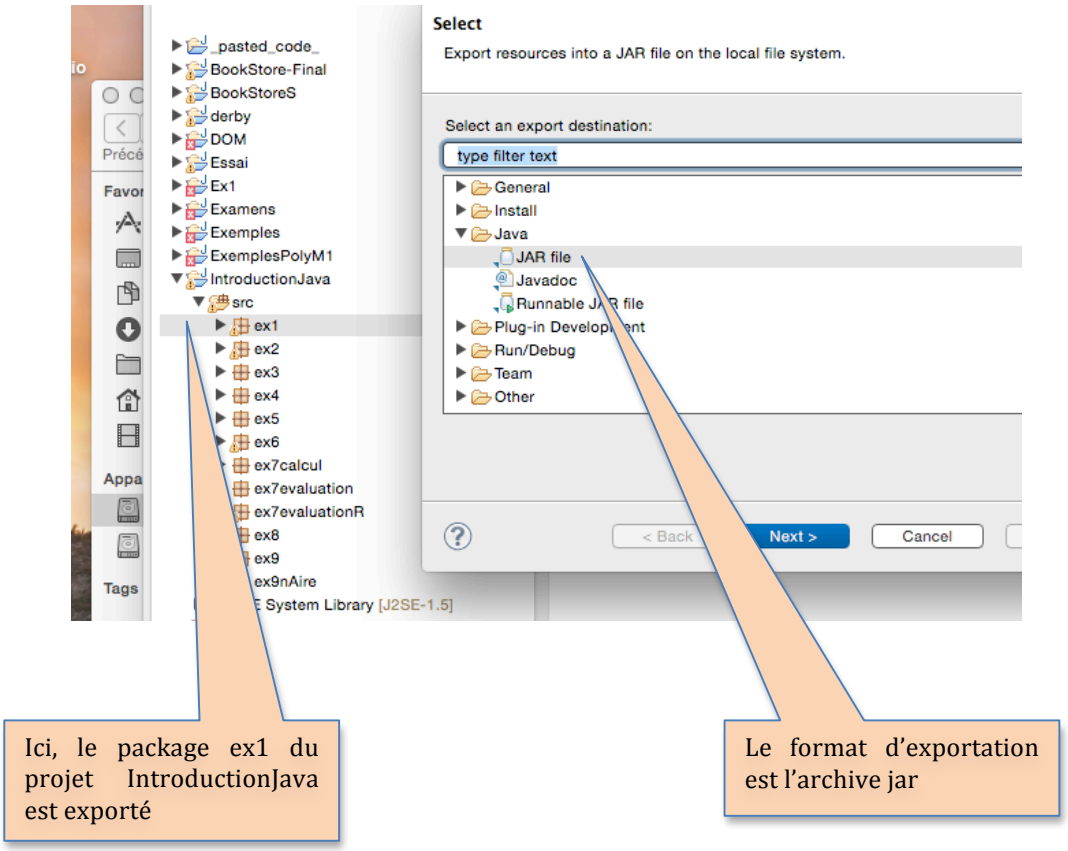
Le tableau `tab` est d'abord déclaré puis crée, il est aussi possible de faire les 2 opérations en une seule instruction (cas des tableaux suivants). L'initialisation du tableau peut se faire à la création; dans le cas d'une matrice, les éléments sont introduits ligne par ligne.
L'attribut `length` permet de récupérer la longueur de l'objet tableau (syntaxe: `objet.attribut`), `mat.length` renvoie le nombre de lignes, `mat.length[1]` renvoie le nombre d'éléments de la deuxième ligne.

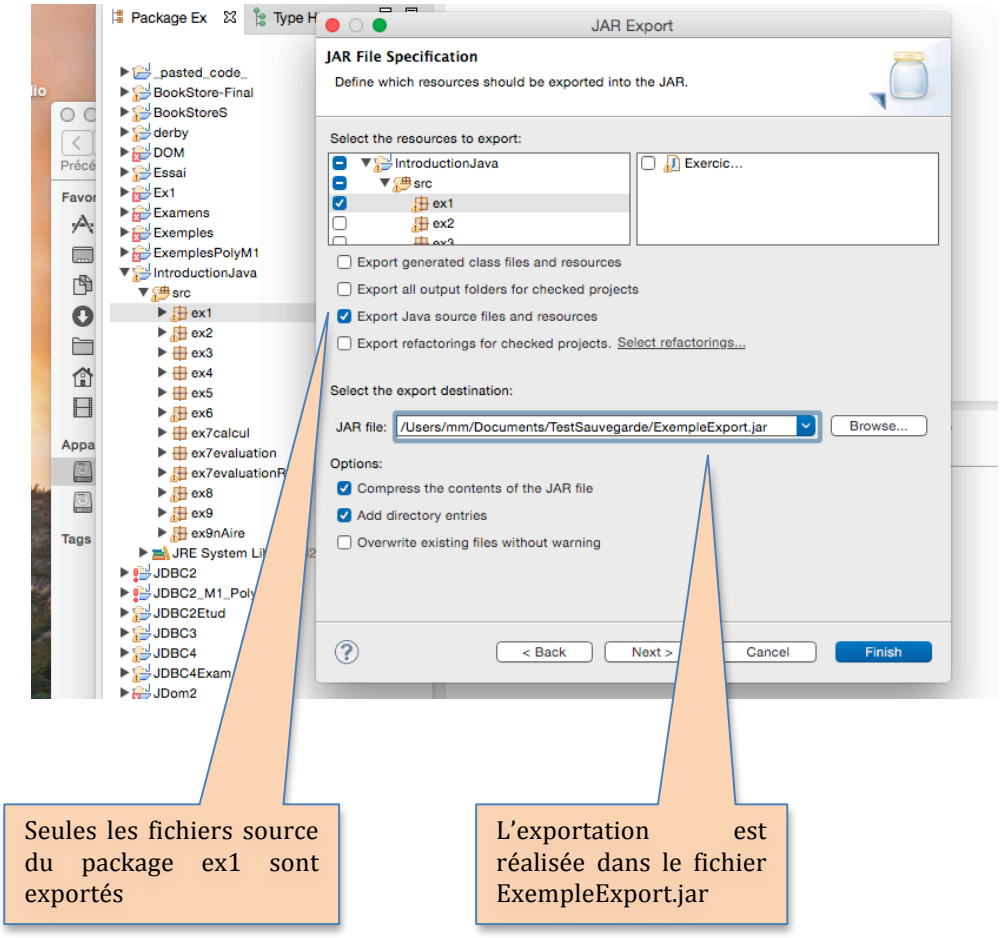
Importer / Exporter un Projet Java (au format jar)

Sous Eclipse une application Java est un ensemble de classes rassemblées dans un ou plusieurs packages (si aucun package n'est précisé, les classes sont placées dans le package par défaut). Le ou les packages sont eux-mêmes rassemblés dans un projet. Sauvegarder une application Java consiste soit à sauvegarder un projet et tout ou partie de ses packages, soit à sauvegarder un package et ses classes (ou une partie de celles-ci). Le résultat est un ensemble de fichiers compressés dans un fichier d'extension `.jar` (Java Archive).

Exportation :

Elle peut être réalisée par le menu principal d'Eclipse (File / Export) ou à l'aide d'un clic droit sur le projet ou le package à exporter. Dans les deux cas on choisira dans le menu déroulant « Jar File » et on précisera le répertoire destiné à recevoir l'archive. Après avoir choisi le projet ou le package à sauvegarder, on pourra préciser les éléments à sauvegarder en cochant / décochant l'item correspondant dans l'arborescence.

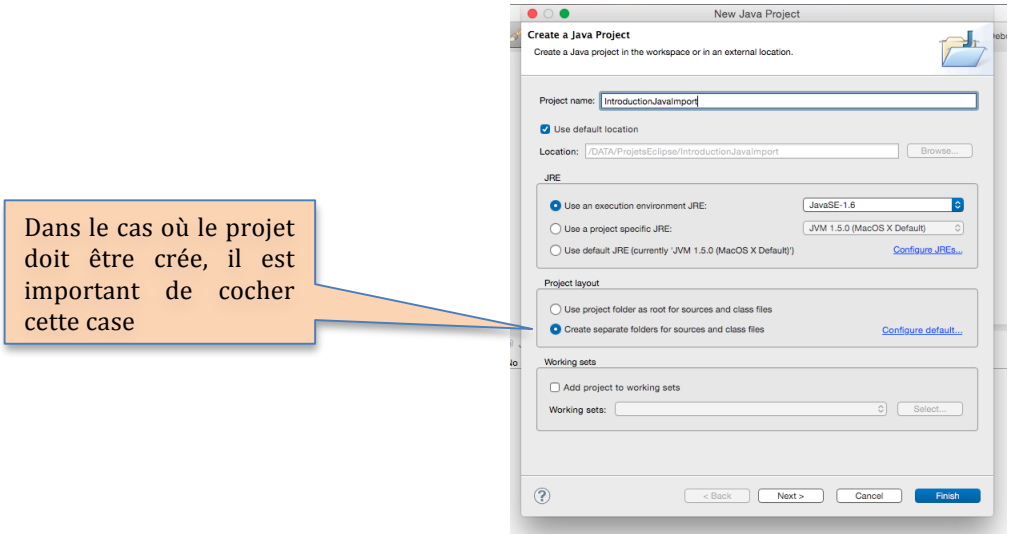




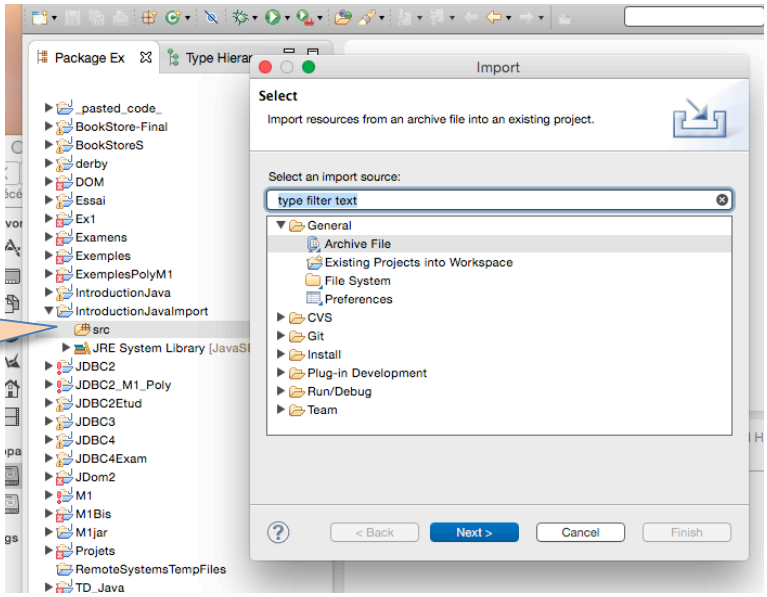
Importation :

Deux cas sont à prévoir selon que le projet recevant l'import existe ou non (si le projet est à créer, on cochera l'option permettant de placer les codes source dans un sous-répertoire séparé). L'importation peut être réalisée à partir du menu principal (File / Import) ou à l'aide d'un clic droit sur le répertoire « src » du projet déjà crée ou à créer.

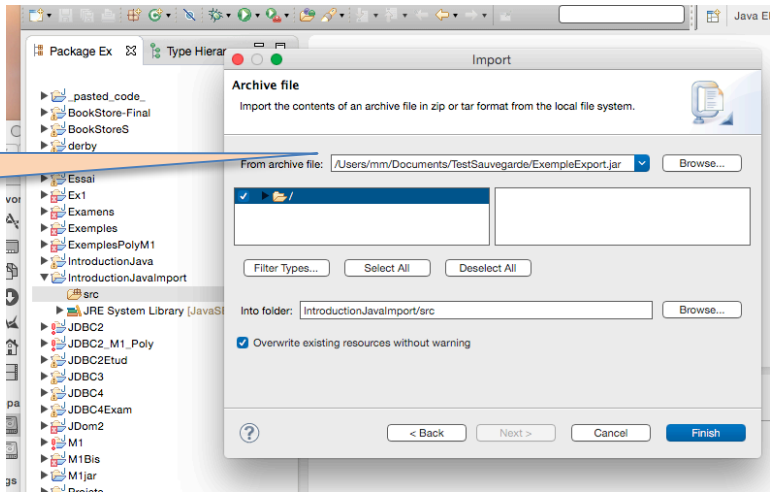
ATTENTION ! Dans l'exemple suivant, le package `ex1` précédemment exporté dans `ExempleExport.jar` est réintroduit dans un nouveau projet `IntroductionJavaImport`.



Ajout d'une archive de type jar dans le projet nouvellement crée IntroductionJavaExport
Attention ! L'ajout doit se faire dans le répertoire src



Sélection de l'archive à importer :
TestSauvegardeExempleExport



Le package ex1 et sa classe Exercice1 ont été correctement importés