PREMIERS PAS EN JAVA

CONTEXTE GÉNÉRAL

UN PEU D'HISTOIRE

Java a été développé dans les années 1990 par des ingénieurs de Sun Microsystems insatisfaits du langage C++, pour les systèmes embarqués :

- Gestion de la mémoire souvent source d'erreur (pointeurs)
- Nécessité de nettoyer soi-même la mémoire en désallouant explicitement les objets : pas de ramasse-miettes (garbage collector en anglais)
- Nom original : oak (chêne)

- 1995 : présentation officielle sous le nom de Java
- Exécution de code dans des pages web au moyen “d’applets”
- Nécessité d’avoir du code portable, compatible avec n’importe quel OS
- Utilisation de la JVM

- Depuis 2000 : sortie d’une version majeure de Java tous les deux ans
- 2009 : rachat de Sun Microsystems par Oracle
- Employé largement dans le développement des premières App. Android
- Langage le plus “populaire” selon l’indice Tiobe…

PRINCIPES DE JAVA

Compilation du code une fois pour toute !

- Fichiers sources : " *.java "
- Après compilation : " *.class " :
  - bytecode **indépendant** du système d'exploitation
  - Destiné à être exécuté par la **JVM** (Java Virtual Machine)
- Une fonction **principale** pour l'exécution : **main**

JAVA : LES PROMESSES

- Déploiement **facile** des applications :
  - Le bytecode généré fonctionne sur tout OS,
  - Une seule compilation / archivage avant la distribution
- Pas de gestion compliquée de la mémoire
- Paradigme objet facile à appréhender (pas d’héritage multiple)

OBJECTIFS DE CETTE SESSION :

- Introduction aux différents **types** en Java
- Introduction aux opérations de base
- Introduction aux instructions de **flow control**
- Introduction à la fonction main

LES TYPES JAVA

TYPES DE BASE

Java est un langage fortement typé. Presque tous les types sont objets.

À l’exception de 8 types de base :

- **byte, short, int, long**
- **float, double**
- **boolean**
- **char**

LES DIFFÉRENTS ENTIERS

- **byte** : entier signé sur 8 bits $[-128, 127]$
- **short** : entier signé sur 16 bits $[-32768, 32767]$
- **int** : entier signé sur 32 bits $[-2^{31}, 2^{31}-1]$
- **long** : entier signé sur 64 bits $[-2^{63}, 2^{63}-1]$

LES ENTIERS LITTÉRAUX

- Par défaut des **int** (ex. : 1, 2, 3,…)
- **byte**, **short**, **int** et **long** peuvent être initialisés avec un entier littéral
- Les **long** avec des valeurs supérieures à 2^{31} peuvent être initialisés avec des littéraux finissant par “L” (ex. : 1000000000L)
- Possibilité d’ajouter des underscores "_" pour la lisibilité (ex. : 1_100)

À ESSAYER DANS L'INTERPRÉTEUR BLUEJ

Note : sans point-virgule, on récupère directement le résultat

```
1_000_000 // On va afficher l'entier 1 million
int i = 1_000_000 ;
10_000_000_000 // On essaye d'afficher 10 milliards (> 2^31)
10_000_000_000L
int j = 10_000_000_000L // Que passe-t-il ?
long k = 10_000_000_000L
byte b = 128 ; // On tente d'affecter 128 à un byte
byte b2 = 127 ;
b2 = b2++ ;  // Que se passe-t-il ?
b2
```

LES DIFFÉRENTES BASES D'ENTIERS

```
// Le nombre 26 en décimal
int decVal = 26;
//  Le nombre 26, en hexadécimal
int hexVal = 0x1a;
// Le nombre 26, en binaire
int binVal = 0b11010;
```

LES FLOTTANTS

- **float** : Nombres flottants simple précision codés sur 32 bits
- **double** : Nombres flottants double précision codés sur 64 bits.

Exemples :

```
double a = 1 ;
double b = 1.0 ;
double c = 1.3e3 ;
```

ATTENTION AUX EXPRESSIONS LITTÉRALES

À tester dans BlueJ…

```
double a = 3 ;
double b = 2 ;
double c = a/b ;

double d = 3/2 ; // Que se passe-t-il ?
```

LES BOOLÉENS

- **boolean** : ne peut valoir que **true** or **false**
- “*Use this data type for simple flags that track true/false conditions. This data type represents one bit of information, but its "size" isn’t something that’s precisely defined.*”

LES CARACTÈRES

- **char** : caractère unicode codé sur 16 bits
- Va de *u0000* à *uFFFF*

Exemple dans BlueJ :

```
  char a = 0 ;
  a
  char capitalC = 'C' ;
  C
  (int)capitalC  // Conversion de capitalC en entier (code
unicode)
```

LE PENDANT OBJET DES ENTIERS ET FLOTTANTS

Il s’agit des classes **Byte**, **Short**, **Integer**, **Long**, **Float**, **Double**

Exemple :

```
a = new Integer(3) ;

Integer (4) // Puis exporter dans le "Object bench"
```

EXERCICE

- Essayer de voir avec BlueJ le nom du champ qui contient la valeur d’un entier dans **Integer**.
- Essayer de lire cette valeur à l’aide de l’interpréteur
- Trouver la méthode permettant de réaliser cette opération.

LES TABLEAUX

Les tableaux (**array**) permettent de stocker un nombre **connu** d'objets en mémoire. Ils peuvent être de n'importe quel type.

Déclaration :

```
TYPE[] tableau ; // TYPE peut être n'importe quoi (int,
double ou une classe)
```

Initialisation :

```
tableau = new TYPE[N] ; // N est un entier
```

Accès :

```
tableau[i] ; // i < N
```

Une autre façon d’initialiser un tableau lorsqu’on connaît les éléments :

```
int[] tableauInt= {3,4,5,6} ;
```

Les tableaux peuvent être modifiés directement :

```
tableauInt[0] = 0 ;
```

Comment connaître la taille d’un tableau ?

- Rappel : le type d’un tableau d’entier est int[] : la taille n’est pas mentionnée.
- **length** : attribut qui stocke la longueur du tableau. Accessible de façon publique. Il s’agit d’une valeur immuable pour un tableau.

```
System.out.println(tableauInt.length) ; // afficher la
taille dans le terminal
```

Il est possible de créer des tableaux sur plusieurs dimensions. Il s'agit alors d'un tableau de tableau.

```
int[][] tableau2dim = {{1 , 2 , 3},{4 , 5 , 6}} ;
```

Exercice :

- Que vaut alors **tableau2dim.length**?
- Vérifier avec l'interpréteur BlueJ
- Que vaut **tableau2dim[1][0]** ?
- Vérifier avec l'interpréteur BlueJ

LA CLASSE STRING

La classe **String** permet de manipuler les chaînes de caractère.

```
String chaine = "Hello World"; // Déclaration et
initialisation
```

MÉTHODES UTILES…

- *int* **length**() : donne la longueur de la chaîne
- *char* **charAt**(*int* i) : retourne la caractère à l'emplacement i
- *String* **concat**(*String* str) : retourne une **nouvelle** chaîne correspondant à la concaténation de l'instance en cours et de l'argument str.

Les chaînes de caractères Java sont immuables :

- Il n’existe aucun moyen de modifier le contenu d’une chaîne.
- L’opération **chaine = "Hello World bis"** revient à placer une **nouvelle** chaîne de caractères dans la variable chaine.
- Autre exemple : la méthode **concat** ne modifie pas la chaîne initiale.

Avec l’interpréteur BlueJ

```
String s1 = " Hello " ;
s1.at(1) = 'B' ; // Voir ce que cela donne

s1.concat("World")  // Sans le point-virgule

String s2 = s1 + " World " ;
s2 // sans le point-virgule

" World ".replace("Wor","Bo")
```

LES INSTRUCTIONS DE BASE EN JAVA

DÉCLARATION / AFFECTATION

- Déclaration : fournir le nom du type et de la variable

```
String s ;
```

- Affectation : avec le signe =

```
s = "Hello" ;
s = new String("Hello"); // Création de l'objet String
avec new
```

- Combinaison déclaration/affectation :

```
String s = "Hello";
```

OPÉRATIONS MATHÉMATIQUES

- Opérations mathématiques de base **+** , **-** , ***** , **/** (avec les priorités mathématiques habituelles)
- Opérations d'incrémentation/décrémentation **++/--**

```
i ++ ; // Comparer avec ++ i ;
```

- Reste de la division entière **%**

```
int i = 11 % 3 ; // i vaut 2
```

TESTS ET LOGIQUE BOOLÉENNE

- Test d'égalité == ou de non égalité !=

```
if (i==3) ...
```

- Tests comparatifs <=, <, >= et >

```
if (i>=3) ...
```

- Opération de négation !

```
Boolean a = !true ; // a = false donc...
```

- Le ET et OU logique : && et ||

```
Boolean b = true && false ; // false !
```

OPÉRATEUR TERNAIRE ?

Si la condition vaut **true**, alors on retourne val1, sinon on retourne val2.

```
condition ? val1 : val2 ;
```

Exemple

```
int note = 15 ;
char grade = (note >= 16) ? 'A':'B';
```

EXCERCICE

Écrire en **une instruction** une fonction qui prend en entrée une note entre 0 et 20 et qui renvoie la lettre associée (**char**) en fonction de la répartition suivante :

- [20 , 16] : A]16 , 14] : B
-]14 , 11] : C]11, 8] : D
-]8 , 5] : E]5, 0] : F

LE FLOW CONTROL EN JAVA

Il s’agit ici de définir l’ordre d’exécution des instructions.

Par défaut, au sein d’une méthode, les instructions sont exécutées **les une après les autres**.

LE MOT-CLÉ RETURN

Le mot-clé **return** permet d'interrompre définitivement l'exécution d'une méthode et de retourner la valeur précisée après le mot clé.

```
int renvoie1(){
  return 1;
}
```

Ce mot clé est obligatoire pour les méthodes qui retournent autre chose que *void*

LES INSTRUCTIONS IF/ELSE

```
instruct1 ;
if (x == 4)
{
  instruct2 ;
}else
{
  instruct3 ;
}
instruct4 ;
```

- Si $x = 4$, on aura instruct1 → instruct2→ instruct4
- Si $x \neq 4$, on aura instruct1 → instruct3 → instruct4
- On peut également avoir **if** sans **else**

LES BOUCLES FOR

```
for (initialisation ; conditionFin ; increment){
  instructions;
}
instructionsSuivante ;
```

- Réaliser des opérations un nombre défini de fois
- Parcourir un tableau / une liste
- Une fois la condition de fin réalisée, **instructionsSuivante** est exécutée

```
for (int i=0 ; i < = 10 ; i++){
  System.out.println("On affiche le nombre "+i);
}
System.out.println("On a compté jusqu'à 10")


for (int i=10 ; i > = 10 ; i--){
  System.out.println("On affiche le nombre "+i);
}
System.out.println("Fin du compte à rebours !")


for ( ; ;){
  // Boucle infinie
}
```

LES BOUCLES WHILE

```
while (expressionTest) {
  instructions;
}
instructionsSuivantes
```

LES BOUCLES DO WHILE

```
do {
  instructions;
}while (expressionTest);
instructionsSuivantes ;
```

- Similaire à **while**
- Mais garantie que le bloc **instructions** est exécuté **au moins une fois**.
- Une fois que **expressionTest** est faux, **instructionsSuivante** est exécutée

LE MOT CLÉ BREAK

Permet de sortir d'un bloc d'instruction **for**, **while** ou **do while** prématurément et d'exécuter les instructions suivantes.

```
String chaine = "Hello World" ;
// Recherche de la présence du caractère 'W'

boolean wPresent = false ;
for (int i = 0 ; i< chaine.length() ; i++){
  if (chaine.charAt(i) == 'W'){
    wPresent = true ;
    break ; // Il n'est plus utile de continuer le for
  }
}
```

LE MOT CLÉ CONTINUE

Permet de “**sauter**” l’itération courante d’un bloc d’instruction **for**, **while** ou **do while**.

```
String chaine = "Hello world" ;
// Comptage du nombre de 'l'

int nb = 0 ;
for (int i = 0 ; i< chaine.length() ; i++){
    if (chaine.charAt(i)!='l')
      continue ; // On passe à i+1

    // On traite le caractère
    nb++;
}
```

L’INSTRUCTION SWITCH

Permet de placer le “control flow” à un endroit spécifique en fonction de la valeur d’une variable parmi un ensemble donné :

```
switch(variable){
  case valeur1 : instr1 ; instr2 ; //...
  case valeur2 : instr3 ; instr4 ; //...
  case valeur3 : instr5 ; instr6 ; //...
}
```

Dès qu’une des conditions est vérifiée, le code exécute **toutes les instructions suivantes**

Si variable = valeur2 alors instr3 → instr4 → instr5 → instr6

EXERCICE

Créer une classe TestSwitch implémentant cette la fonction suivante :

- prototype : *String* **getDay**(*int* i)
- Retourne la chaîne de caractères associées au $i^{ème}$ jour de la semaine
- Robuste à un utilisateur malicieux

Cette version ne compilera pas (dayString potentiellement non initialisée) et est sémantiquement incorrecte (manque **break**)

```
public String getDayBAD(int dayNumber){
  String dayString ;
  switch(dayNumber){
    case 1 : dayString = "Monday" ;
    case 2 : dayString = "Tuesday";
    case 3 : dayString = "Wednesday";
    case 4 : dayString = "Thursday";
    case 5 : dayString = "Friday";
    case 6 : dayString = "Saturday";
    case 7 : dayString = "Sunday";
  }
  return dayString ;
}
```

```
public String getDay(int dayNumber){
   String dayString ;
   // Plus besoin d'initialiser grâce au default.
   switch(dayNumber){
    case 1 : dayString = "Monday" ; break ;
    case 2 : dayString = "Tuesday"; break ;
    case 3 : dayString = "Wednesday"; break ;
    case 4 : dayString = "Thursday"; break ;
    case 5 : dayString = "Friday"; break ;
    case 6 : dayString = "Saturday"; break ;
    case 7 : dayString = "Sunday"; break ;
    default: dayString = "I am Groot";
  }
  return dayString ;
}
```

- Penser aux **break** et **default**
- Ne fonctionne qu’avec les **int** (et dérivés) et les **String**. Les valeurs testées doivent être constantes

LA STRUCTURE D'UNE APPLICATION JAVA

RETOUR SUR LE TUTORIEL BLUEJ

- Deux fichiers sources .java
- Les versions compilées sont les .class
- *Cat.java* décrit le fonctionnement de la classe **Cat**
- Comment faire pour **créer un programme exécutable** ?

LA FONCTION MAIN

Il est possible “d’exécuter” une classe si et seulement si celle-ci contient une fonction **main** dont la signature est la suivante :

```
public static void main(String[] args) ;
```

- **public** : la méthode est publique (peut être appelée depuis une autre classe)
- *(**static** : la méthode est statique. C'est une méthode de classe qui ne nécessite pas d'instance de l'objet.)*
- **void** : la méthode ne retourne rien.
- **String**[] **args** : l'argument de main est un tableau de String.

POURQUOI STRING[] ARGS?

- Identification des arguments lors d'une commande textuelle dans un terminal

```
$ ls -l *.java      (unix)
$ dir *.java        (windows)
```

Liste tous les fichiers avec l'extension java et les présente sous forme de liste

RÉCUPÉRER LES ARGUMENTS

```
$ ls -l *.java
```

RÈGLE D'USAGE POUR MAIN

En général, il est préférable de définir une classe particulière qui contient le Main. Il est rarement approprié de définir cette fonction dans une classe normale.

POUR EXÉCUTER

Se mettre dans le répertoire contenant le fichier .class (ex. MainClass.class) issu de la compilation à l'aide de la commande **cd**.

```
$ java MainClass argument1 argument2....
```

UNE CLASSE MAINCLASS POUR LES CHATS

- Arguments autorisés : listen, happy?, feed, adopt_another
- Être robuste à un utilisateur maladroit…
- Ajouter une méthode publique **getName()** à **Cat** pour que Main puisse accéder au nom du chat

```
  $ java MainClass
  You adopt a tabby cat named Bob

  $ java MainClass happy? listen feed listen adopt_another
happy? mauvaisArgument feed happy?
  You adopt a white cat named Cole
  Cole is not happy
  Cole meows at you.
  Cole purrs.
  You abandon Cole
  You adopt a white cat named Marmalade
  Marmalade is not happy
  My name is Groot!
  Marmalade is happy
```

Note : opérateur new pour initialiser un objet (cf tuto)