Programmation Objet
Java – Partie Impérative

Michail Lampis
michail.lampis@dauphine.fr

Informations Pratiques

- Page web :

  http://www.lamsade.dauphine.fr/~mlampis/Objet/
- Cours : Mardi, 13h45-15h15
- TD/TPs : 3 groupes
  - Jeudi 15h30-18h45

    Responsables : Michail Lampis (TD), Diana Jlailaty (TP)
  - Vendredi 13h45-17h00 (2 groupes)

    Responsable : Michail Lampis (TD), Geovanni Rizk (TP), George Butler (TD+TP)

Informations Pratiques

- Notation :
  - 60 % examen final
  - 20 % examen partiel
  - 20 % CC == Exos de programmation, présentés pendants les TPs
    - ~1 exo / 2 semaines
- Note finale : Max(EF, 0.6*EF+0.2*P+0.2*CC)

Plan du Cours

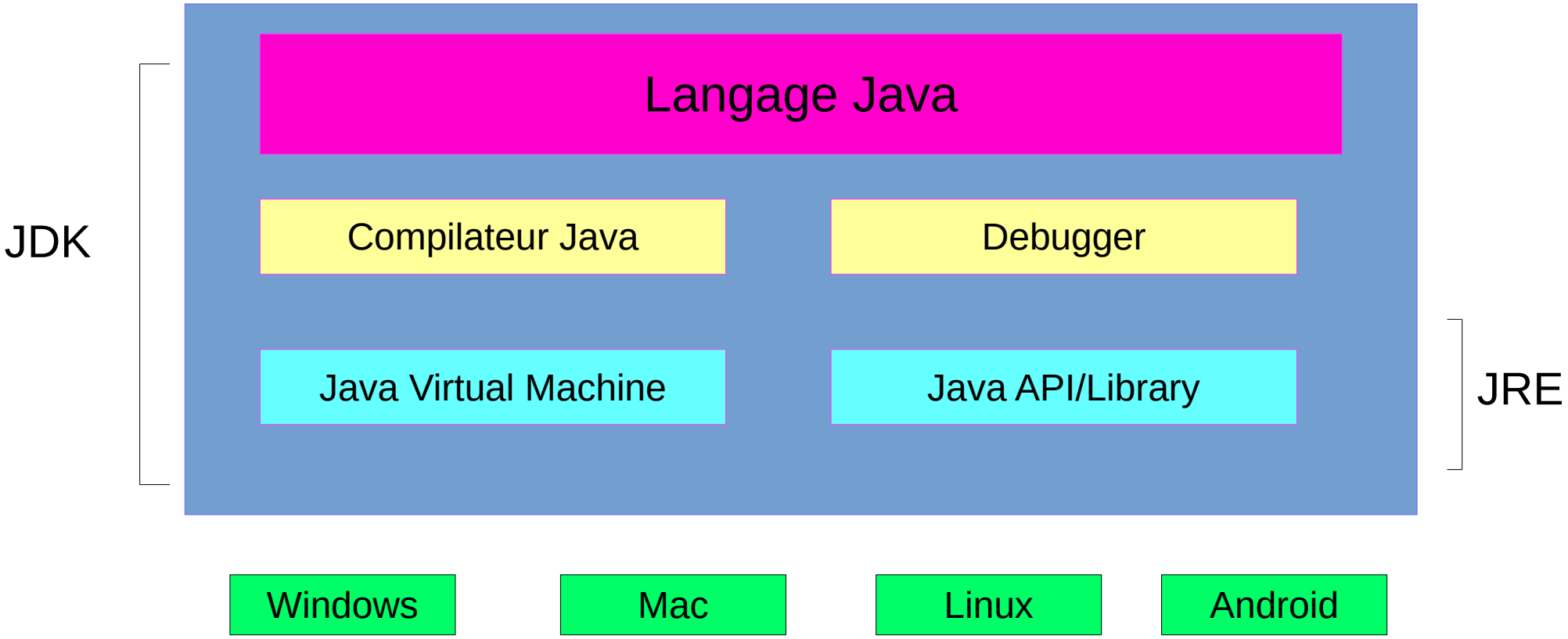
- Programmation Objet + Java

1) Syntaxe de Java
2) Concepts de l'OOP (Object-Oriented Programming)
    1) Encapsulation
    2) Héritage
    3) Polymorphisme
3) ADTs (Abstract Data Types)
4) Interfaces
5) Classes Génériques
6) Exceptions
7) Expressions Lambda
8) ...

Java - Introduction

- Plus qu'un langage de programmation, une vrai plateforme:
  - Un langage
  - Une bibliothèque très vaste (Java API*)
  - Un environnement d'execution portable (Java VM + Java Plugins)
- Version actuelle : Java 11

Architecture de Java - JVM



JDK, JRE, IDE, …

- JDK = Java Development Kit
  - Logiciel pour programmer en Java
  - Notamment : Compilateur

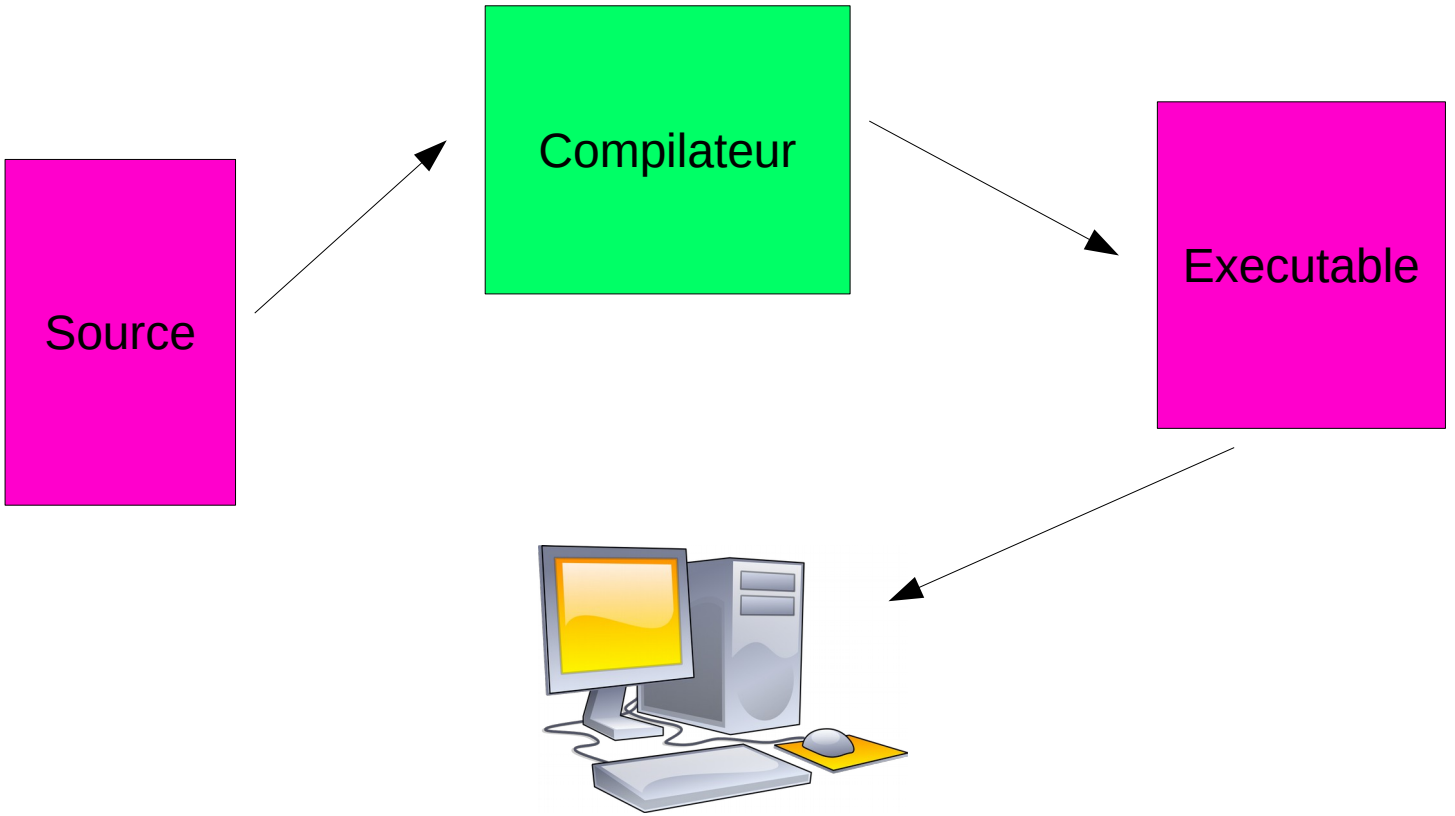
    http://www.oracle.com/technetwork/java/javase/downloads/
- JRE = Java Runtime Environment
  - Logiciel nécessaire pour l'exécution de programmes Java
- IDE = Integrated Development Environment
  - Logiciel qui facilite le développement (pas obligatoire)
  - Exemple : Eclipse

Caractéristiques de Java

- Free and Open Source
- Portable, Indépendante de platforme
- Syntaxe à la C
- **Orienté Objet**
- Programmer-friendly (par rapport à C++)
  - Pas de pointeurs
  - Garbage Collection
- Vaste Librairie

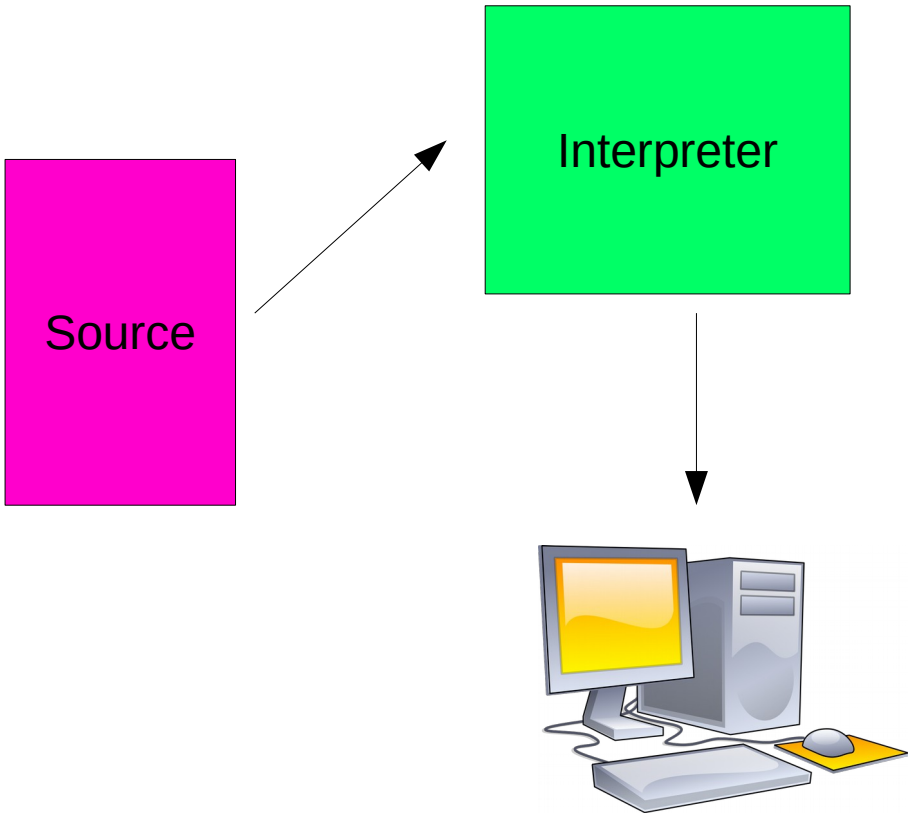
Compilation

- Compilation Classique (ex : C/C++)



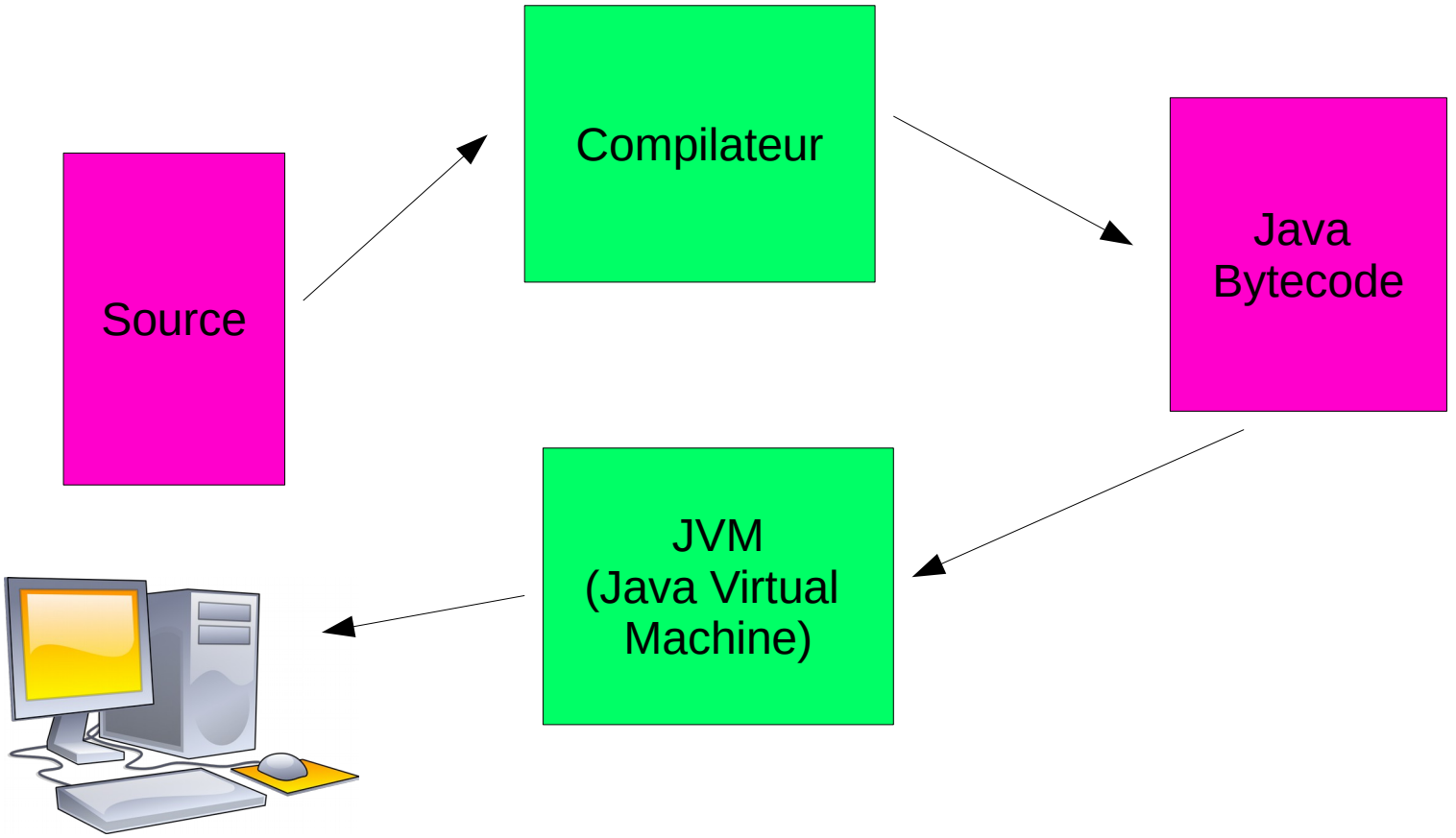
Compilation/Interprétation

- Interprétation (ex : Python)



Compilation Bytecode

- Compilation Java



Java Virtual Machine

- Avantage : Portabilité
  - Le Java bytecode peut être exécuté où il existe une JVM, sans récompilation
- Désavantage : Efficacité
  - Exécution plus lente par rapport à un exécutable native.

Ma première application

Fichier : Hello.java

```
/* This is a comment */
class Hello {
	public static void main(String [ ] args){
		System.out.println(‘’Hello !”) ;
		//This is also a comment
	}
}
```

Compilation, Exécution

- Pour compiler le programme

  javac Hello.java

  - Cette commande appelle le compilateur (javac)
- Pour exécuter

  java Hello

  - Appelle la JVM
- Attention : le fichier doit avoir le même nom que la classe principale (Hello)

Programmation Impérative

- Java est un langage OOP
  - Cependant, on va commencer avec sa partie « classique », sans utiliser les classes et les objets
- Connaissances de base :
  - Variables/Types primitifs
  - Fonctions (statiques)
  - Boucles, Conditionnels
  - Tableaux
  - Récursivité
  - I/O

Variables

- Déclaration d'une variable : Type+nom
  - Ex : **int x ;**
- 8 types primitifs :
  - **int, long, short, byte** : entiers
  - **float, double** : décimaux
  - **char** : caractère
  - **boolean** : true or false

Affectations

- Une variable prend une valeur avec l'opérateur =

```
int x = 2 ;
int y,z ;
y = x+5 ;
z = y = 7 ;
```

  - (y=7) est une expression de valeur 7

Affectations

- L'opérateur = fait une copie de la valeur à droite, s'il s'agit d'une valeur primitive

```
int x = 5;

int y = x ;

x++ ; //x = x+1 ;

System.out.println(y) ; //affiche quoi?
```

- **Attn :** ce n'est pas le cas pour les objets ! (on le verra plus tard)

Opérations Élémentaires

- Pour les int (long,short,...)
  - +,-,*,/, %. Attention : / est division entière,

    5/2 == 2
  - ++,--
    - Ex : **int x=3 ; x++ ; x== ??**
    - **++x** vs. **x++**? Précédence...
- Comparaisons :
  - ==,!=. Attention : différence entre =et==
  - <,<=,>,>=

Opérations Élémentaires

- Opérations Booléennes

  &&, ||, !: and, or, not

- Ternary Operator

  A ? B : C

  - Ex :

    System.out.println((5>3) ? “Bla” :”No”) ;

Control Flow

- If statements

  if(« condition »){ stmt;stmt ;…}

  *[***else {stmt;stmt ;…}** *]*

  - Ex :

    int x = 3 ;

    if(x==5) System.out.println(“five”) ;

    else System.out.println(“not five”) ;

- Attn : if(x==5) vs if(x=5)

Dangling else

```
int x = 4 ;
if(x>5)
    if(x>8)
        System.out.println(“>8”) ;
else
    System.out.println(“<5”) ;
```

- Affiche quoi ?

Dangling else

```
int x = 4 ;
if(x>5)
    {if(x>8)
        System.out.println(“>8”) ;
else
    System.out.println(“<5”) ;}
```

  - N’affiche rien !

- else appartient à if le plus proche
- Utiliser {} pour désambiguïser !

Boucle for

for(<init>;<condition>;<increment>){

stmt ; stmt ;... }

- Étape 1 : expression <init> est évalué
- Étape 2 : lorsque la <condition> est vraie
  - On exécute les stmts
  - Puis on évalue l'expression <increment>
- Si la <condition> reste vrai, retour à étape 2

Boucle for

```
for(int i=1; i<11; i++){
        System.out.println("Count is: " + i);
}
```

- Affichage :

  Count is 1

  ...

  Count is 10

- Attn : déclaration de i dans la boucle

Boucle while

while(<condition>){stmt ; stmt ;...}

- Équivalence avec

  for(; <condition>;){stmt ;...}

  - NB : quelques expression de **for/while** peuvent être vides
  - Ex : **for(;;) ; //**Infinite loop !!

Boucle do while

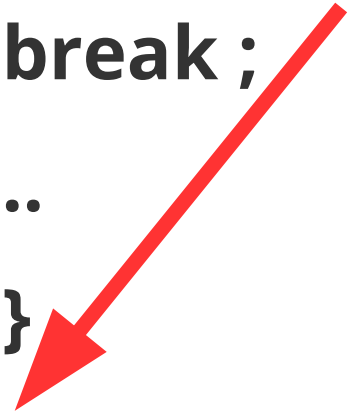
do{

stmt1 ; stmt2 ;... }while(<condition>)

- <condition> est évaluée après les stmts
  - >=1 exécution

Break et continue

```
for(.. ;..;..){
    ..
    ..
    break ;
    ..
}
```



Break et continue

```
for(.. ;..;..){
    ..
    ..
    continue ;
    ..
    }
```

Les Fonctions

- Une fonction (méthode) est une partie indépendante de notre programme.
- Comme une fonction mathématique elle prend des paramètres et retourne une valeur
  - Elle peut aussi avoir de side-effects
- Syntaxe : <type> <name>(<params>..){..}
- Mot clé : **return**

Fonctions

```
int max(int x, int y){
    if(x>y) return x ;
    else return y ;
    System.out.println("See you never !") ;
}
```

- Éléments :
  - On a défini type de paramètres, return
  - return termine la fonction

Fonctions statiques

- En Java, les fonctions sont normalement associés avec un objet (OOP) – méthodes
- Les méthodes ont en plus des limitations d'accès.
  - On va en parler plus tard
- Pour définir une fonction utilisable partout et pas associé a un objet, on écrit

  public static

Exemple

```
class Hello {
        public static int max(int x, int y){
                return (x>y)?x:y;
        }
        public static void main(String [] args){
                System.out.println(max(3,5));
        }
}
```

Call-by-Value

- Quand il s'agit des types primitifs, une fonction est passé **une copie** de chaque paramètre
- Ex :

```
public static void f(int x){ x++;}
public static void main(String []args){
    int a=2 ; f(a) ; S.o.pln(a) ;//a=2
}
```

Call by Value

- Call by value est bon : on peut passer des expressions compliquées comme paramètres

```
public static void f(int x){ x++;}
public static void main(String []args){
    int a=2 ; f(2*a+3) ; S.o.pln(a) ;//a=2
}
```

Récursivité

- Récursivité : quand une fonction f appelle la fonction f
- Attn : il faut bien y avoir un cas de base
  - Sinon, infinite loop !
- Exemple : Écrire une fonction qui retourne le plus grand commun diviseur (pgcd) de x,y
  - Rappel : pgcd(x,y) = pgcd(x,y-x), si y>x

pgcd

```
public static int pgcd(int x, int y){
        if(x<y) return pgcd(y,x);
        if(y==0) return x;
        // x>y>0
        return pgcd(x-y,y);
}
public static void main(String [] args){
        System.out.println(pgcd(2*3*5*7,2*7*9));
}
```

Efficacité

- Quelle est l'utilisation de mémoire de la fonction récursive qui calcule le pgcd ?
  - Ex : pgcd(10000,1) ;
- Pile d'appels (function call stack) de taille (dans le pire de cas) max(x,y)
  - Peut-on faire mieux ?

Pgcd avec une boucle

```
public static int pgcd(int x, int y){
    while(x!=y){
        if(x>y) x-=y; //x = x-y
        else y-=x;
    }
    return x;
}
```

Tableaux

- Un tableau est une structure de données qui peut contenir plusieurs éléments du même type.
- On peut accéder à chaque élément en utilisant une indice.
- Syntaxe : pour déclarer un tableau

  T [] myTableau ;

  int [] myIntArray ;

Tableaux

- Ayant déclaré une variable de type tableau, il faut initialiser le tableau :

  int [] myArray = new int [10] ;

  Réserve un tableau de 10 positions

- Pour accéder aux éléments on utilise les []
  - Attn : le premier élément est à position 0

Tableaux

```
int [] a = new int [10];
a[0] = 5;
a[1] = 3;
a[2] = 2;
a[3] = 4;
System.out.println(a[1]);
System.out.println(a[a[1]]);
```

- Affichage ?

Out of Bounds

- Attn : il faut que l'indice utilisé soit toujours entre 0 et taille-1.

  int [] a = new int [10] ;

  a[10] = 10 ;

- Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10
- Le compilateur ne peut pas vérifier cette condition (pourquoi?)

Parcourir un tableau

- La taille d'un tableau est donné par l'attribut **length**
- Ex :

```
int [] a = new int [10];
for(int i=0; i<a.length; i++)
        a[i]=2*i;
for(int i=0; i<a.length; i++)
        System.out.println(a[i]);
```

Parcourir un tableau

- Version plus moderne de la boucle for

```
//initialisation du tableau
int[] numbers ={1,2,3,4,5,6,7,8,9,10};
for (int item : numbers) { //for each
  System.out.println("Count is: " + item);
}
```

Références

- Attn : une variable de type « Tableau de T » n'est pas une variable primitive
  - Pas de call-by-value. Cette variable est un **référence**
- Ex :

```
int [] a = {1,2,3};

int [] b = a;

b[0] = 7;

for(int i:a) System.out.println(i);
```

Références

- Ex :

```
public static void f(int []x){x[0]++;}
public static void main(String [] args){
    int [] a = {1,2,3};
    f(a);
    for(int i:a) System.out.println(i);
}
```

Valeur Moyenne

- Écrire une fonction qui calcule la valeur moyenne d’un tableau de ints.

Valeur Moyenne

- Écrire une fonction qui calcule la valeur moyenne d'un tableau de ints.

```
public static double avg(int [] a){
        double res = 0.0;
        for(int i:a) res += i;
        return res/a.length;
}
```

Entrée/Sortie

- On va exécuter nos programmes à la console
- → On va communiquer avec l'utilisateur avec le clavier
- Pour afficher un message :

  System.out.println(<string>) ;

  System.out.print(<string>) ;
- ln = new line
- NB : paramètre est automatiquement converti en string

Entrée

- Pour demander à l'utilisateur d'écrire quelque chose avec le clavier

  classe **Scanner**

```
import java.util.Scanner;
class Hello {
    public static void main(String [] args){
        Scanner s = new Scanner ();
        int x;
        System.out.println("Give me a number");
        x = s.nextInt();
        System.out.println("Your number is "+x);
    }
}
```

Scanner

- La classe scanner peut être utilisé pour lire d’autres types de données
- Documentation :

  https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html

Command line parameters

- Rappel :

  public static void main(String [] args)

- args est un tableau qui contient les paramètres donnés quand le programme a été exécuté.

- Ex :

```
class Hello {
    public static void main(String [] args){
        for(String s: args)
            System.out.println("argument: "+ s);
    }
}
```

Command line parameters

- Exécution :

  mlampis@mlampis:~/tmp/java$ java Hello 1 2 as 3

  argument: 1

  argument: 2

  argument: as

  argument: 3

Command line parameters

- Les paramètres du tableau args ont comme type **String**
  - Pour les convertir en int, on peut utiliser la fonction **Integer.parseInt**
  - Ex :

```
public static void main(String [] args){
    for(String s: args){
    System.out.println("argument: "+    (Integer.parseInt(s)+1));
    }
}
```

Command line parameters

- Exécution

  mlampis@mlampis:~/tmp/java$ java Hello 1 2 3

  argument: 2

  argument: 3

  argument: 4