

Module de Java avancé

ESGI 2015 - 2016

Frédéric Baudoin

MODALITÉS ET OBJECTIFS

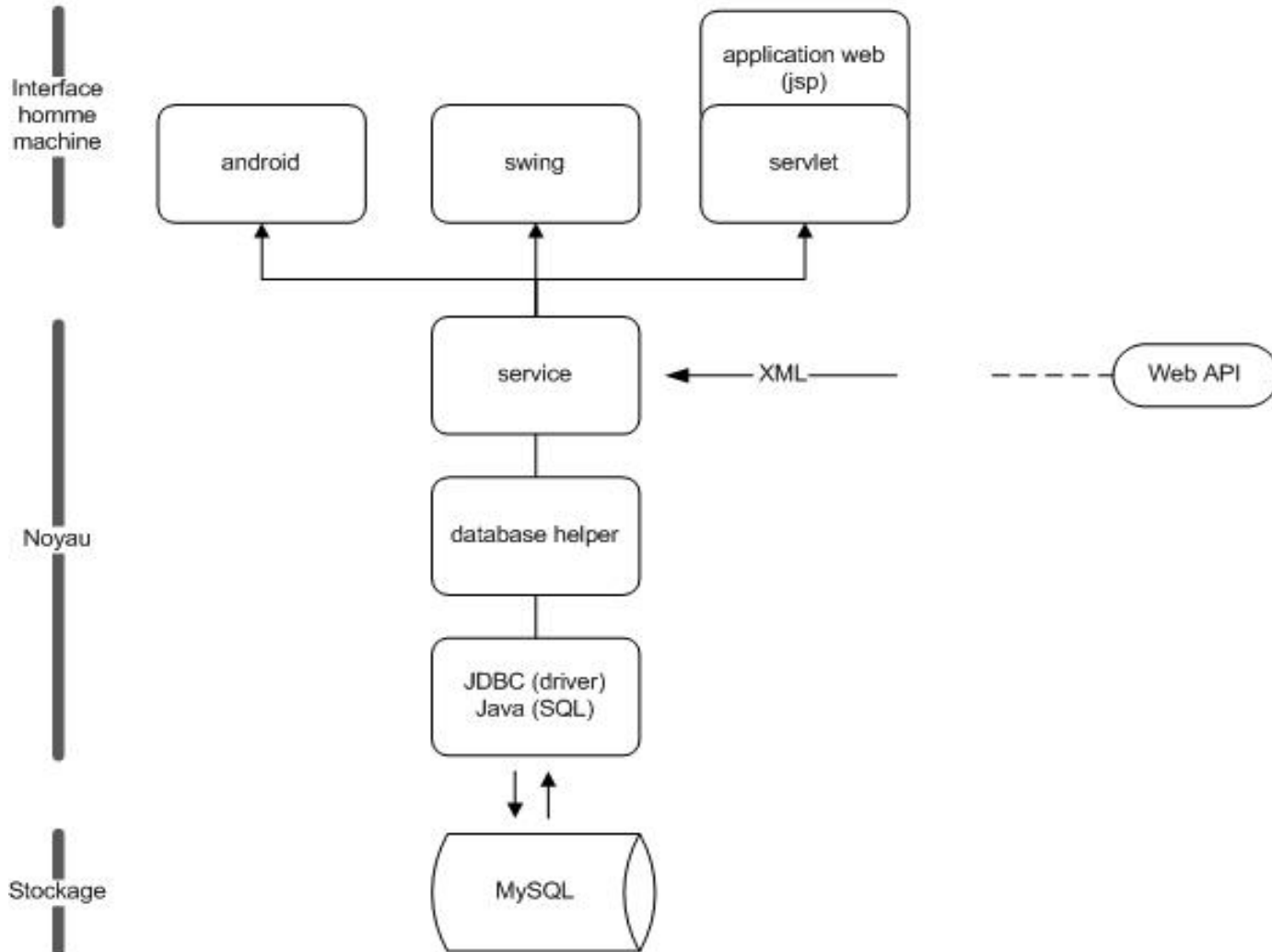
Progression

- Cycle 1 :
 - Environnement de développement
 - Types primitifs, opérateurs, conversion,
 - Variables
 - Structures de contrôle
 - Tableaux
 - Chaînes de caractères
 - Entrées/sorties
 - Programmation orientée objet (POO)
 - Généralités
 - Héritage
 - Classes abstraites et interfaces

Progression

- Cycle 2 :
 - Collections
 - Flux
 - Exceptions
 - Programmation événementielle (Swing)
- Cycle 3 :
 - connexion BDD : JDBC

Architecture des *applications de gestion*



Objectif du cours

- Acquérir les compétences pour développer en Java des **applications de gestion**
 - Interface graphique (Swing)
 - Noyau (design patterns)
 - BDD
 - Fichiers
- A la limite du programme :
 - *XML et requêtes web*
 - *Threads*

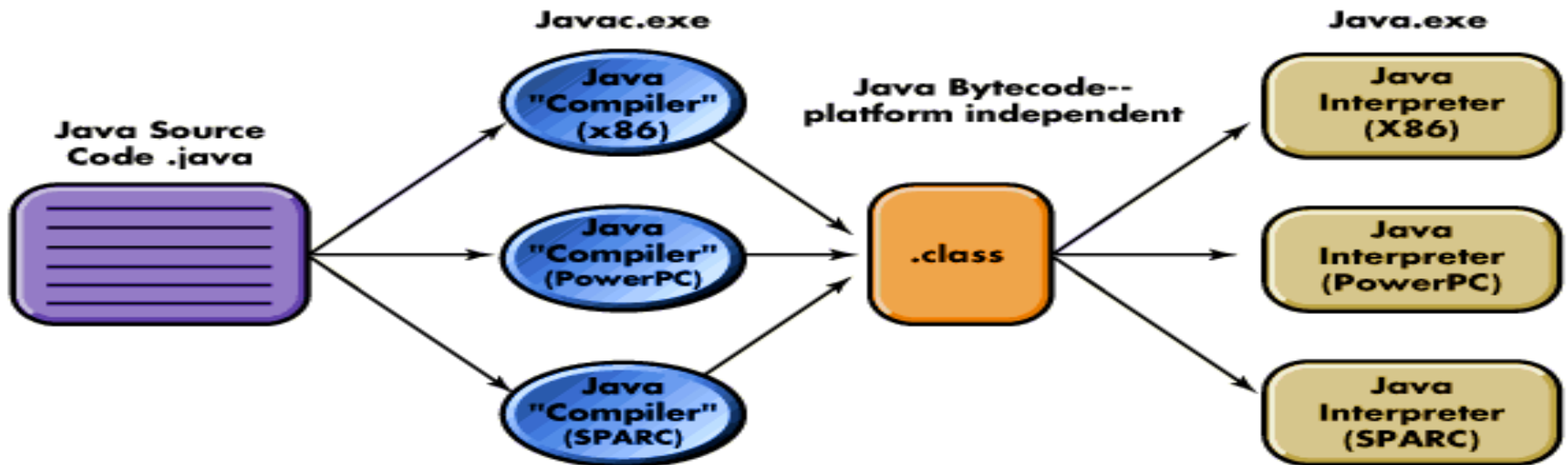


Evaluation

- TP à envoyer à fbaudoin@myges.fr avec comme objet :
 - Java 3DW VALJEAN Jean TP7
- 2 contrôles continus
- 1 projet à réaliser en binôme

Technologie Java

- javac.exe : un compilateur Java,
- java.exe une machine virtuelle Java qui permet d'interpréter le *bytecode java*,
- API java : une bibliothèque de fonctionnalités qui donne la force au langage Java et possède une documentation disponible en ligne ([L'API de java 7](#)).



Révisions

CYCLE 1

Types primitifs

Primitive	Signification	Taille (en octets)	Plage de valeurs acceptée
char	Caractère	2	valeur du jeu de caractères Unicode (65000 caractères)
byte	Entier très court	1	-128 à 127
short	Entier court	2	-32768 à 32767
int	Entier	4	-2 147 483 648 à 2 147 483 647
long	Entier long	8	-9223372036854775808 à 9223372036854775807
float	flottant (réel)	4	$-1.4 \cdot 10^{-45}$ à $3.4 \cdot 10^{38}$
double	flottant double	8	$4.9 \cdot 10^{-324}$ à $1.7 \cdot 10^{308}$
boolean	booléen	1	0 ou 1 (=autre)

Types primitifs et non primitifs

- Types primitifs
 - int, double, float, char, ...
 - variables manipulées par valeur
 - Types non primitifs = types références
 - variables permettant de référencer objets et tableaux
 - variables manipulées "par référence"
 - valeur par défaut *null*
- Les variables passées en argument des méthodes par recopie de leur valeur sont :
- non modifiables : types primitifs
 - modifiables : types référence

Exercise

```
class PetitObjet {  
    private int valeur = 5;  
    int getValeur() { return this.valeur; }  
    void setValeur(int val) { this.valeur = val; }  
}  
  
class EssaiPassage {  
  
    static void travailler(int[] tableau, int n, PetitObjet petit) {  
        tableau[0] = 4;  
        n = 7 ;  
        petit.setValeur(10);  
    }  
  
    public static void main(String[] arg) {  
        int[] table = {1, 2};  
        int n = 3;  
        PetitObjet petit = new PetitObjet();  
        travailler(table, n, petit);  
        System.out.println(table[0] + " " + n + " " + petit.getValeur());  
    }  
}
```

Opérateurs arithmétiques

- Addition +
- Soustraction -
- Multiplication *
- Division /
- Modulo (reste) %

Conversions implicites

- Expression numérique avec différents types
 - Conversion d'ajustement de type suivant la hiérarchie des types numériques :
Int \rightarrow long \rightarrow float \rightarrow double
 - Exemple : $i * l + f$
 - Avec `int i; long l; float f;`
 - Ajustement :
 - i de *int* en *long* pour $*$
 - $i * l$ de *long* en *float* pour $+$
 - $i * l + f$ est un *float*

Conversions implicites

- Promotions numériques
 - Concerne les types exclus des opérateurs arithmétiques :
 - byte, short et char
 - Conversions vers le type *int*

Opérateurs relationnels

- Inférieur <
- Inférieur ou égal <=
- Supérieur >
- Supérieur ou égal >=
- Égalité ==
- Inégalité !=

Opérateurs logiques

- Négation !
 - Et &
 - Ou exclusif ^
 - Ou inclusif |
-
- Et coupe circuit &&
 - Ou coupe circuit ||

Conversions implicites

- Conversion par **affectation**
 - Même principe que pour l'évaluation d'instructions :
byte → short → int → long → float → double
char → int → long → float → double
 - Exemple : (conversion de *int* en *float*)
Int i ; float f ;
f = i + 1

Opérateurs incrémentation et décrémentation

- Post ou pré incrémentation `i++` ou `++i`
 - Suivant la valeur de l'instruction désirée
 - Exemple `System.out.println(t[i++]);`
- Affectation élargie
`n +=1` ou `a*=b`

Conversion explicite (casting)

- Rôle : Force le type des variables
- Est prioritaire sur les autres opérateurs
- Exemple : `int a=5 ; int b=4;`

`(double)(a/b)` vaut 1.0

`(double)a/b` vaut 1.25

`(int)((double)a/b)` vaut 1

Tableaux

Déclaration

```
char[] tableau;   ou  
char tableau[];
```

Création et initialisation

- Opérateur *new*
tableau = new char[3];
tab[0] = 3; tab[1] = 4; tab[2]=1;

Initialiseur

```
int[] tab = {3,4,1};
```

Tableaux

Parcours d'un tableau de *char*

```
for (int i= 0; i < tableau.length; i++)  
    System.out.print(tableau[i] + " ");
```

```
for (char valeur : tableau)  
    System.out.print(valeur + " ");
```

Chaînes de caractères

Type String

- `String nom = new String(« pierre »);`
- `String nom = « pierre »;`

Structures conditionnelles

- **if** (condition réalisée) {
 //liste d'instructions
}
 else {
 //liste d'instructions
 }
- **switch** (expression)
 { **case** *Valeur1* : { liste d'instructions } **break**;
 case *Valeur2* : { liste d'instructions } **break**;
 case *Valeurs...* : { liste d'instructions } **break**;
 default: { liste d'instructions } **break**;
 }

Boucles

- **while** (condition) {
 //liste d'instructions
}
- **do** {
 // liste d'instructions
}
while (condition)
- **for** (initialisation du compteur ; critère d'arrêt ; incrémentation) {
 liste d'instructions
}

Structures de contrôle

- L'instruction *break* : fait sortir de la boucle la plus interne

While (condition){

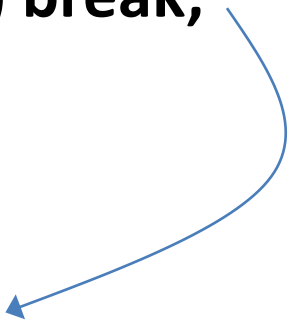
...

if() break;

...

}

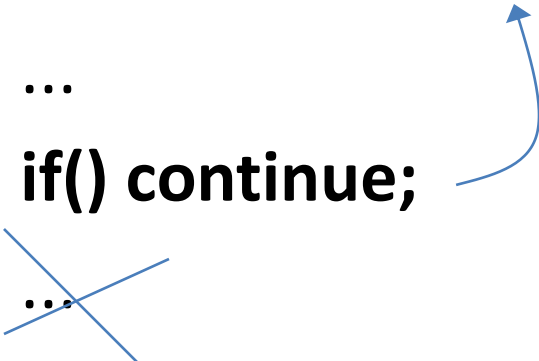
....



Structures de contrôle

- L'instruction *continue* : permet de passer à la répétition suivante

```
For(int i=1;i<4;i++){  
    ...  
    if() continue;  
    ...  
}
```



Sous-fonctions

- Déclarer une fonction

```
public static TYPE_RETOUR nom(TYPE arg1, TYPE arg2,  
    ...) {  
    //INSTRUCTIONS  
}
```

- Appeler une fonction

```
...  
nom(valeur1, valeur2, ...);  
...
```

Hygiène

- Clarté du code
 - Indentation
 - Commentaires
 - Sous fonctions
- Nommage
 - Eloquence des noms utilisés pour les variables
 - Convention
 - pas de symboles exotiques (\$, _)
 - Classe
 - variable, méthode, attribut
 - variableOrdinaire
 - CONSTANTE
 - CONSTANTE_ORDINAIRE
 - com.esgi.3a.java.tp.package