



Cours de Java

Sciences-U Lyon

Java - Introduction

Java - Fondamentaux

Java – Avancé

<http://www.rzo.free.fr>

pierre.parrend@ieee.org



Sommaire



- Java – Introduction
- Java – Fondamentaux
 - Histoire de Java
 - Machine Virtuelle
 - Documentation
 - Qualité logicielle
 - ...



Sommaire



- Java – Fondamentaux
 - ...
 - Rappel d'algorithmie
 - **Langage Java**
 - Performances de Java
 - Conception
- Java - Avancé



Le langage Java

- Sommaire
 - **Formats de données**
 - Premiers Objets
 - Exceptions
 - Structure de programme
 - Input / Output



Le langage Java

- Formats de données
 - **Concepts Objets**
 - Variables
 - Opérateurs
 - Expressions et spécifications
 - Contrôle de flux



Le langage Java

- <http://java.sun.com/docs/books/tutorial/>
- **Concepts Objet**
 - **Objet** : ensemble de variables et méthodes, représentant un objet réel
 - **Message** : moyen de communication des objets entre eux
 - **Classe** : prototype définissant les variables et méthodes communs à un ensemble d'objets



Le langage Java

- Concepts Objet
 - **Héritage** : les classes héritent des caractéristiques des super-classes. C'est un moyen de structurer les programmes
 - **Interface** : ensemble de déclarations de variables et de méthodes.



Le langage Java

- Formats de données
 - Concepts Objets
 - **Variables**
 - Opérateurs
 - Expressions et spécifications
 - Contrôle de flux

Langage Java

- Format des données : variables
 - Déclaration : type nom
 - Exemple :

```
// integers
byte largestByte = Byte.MAX_VALUE;
short largestShort = Short.MAX_VALUE;
int largestInteger = Integer.MAX_VALUE;
long largestLong = Long.MAX_VALUE;

// real numbers
float largestFloat = Float.MAX_VALUE;
double largestDouble = Double.MAX_VALUE;

// other primitive types
char aChar = 'S';
boolean aBoolean = true;
```

Langage Java

- Format des données : variables
 - Types primitifs de données

mot clé	Description	Format
byte	entier d'un octet	8 bits
short	entier court	16 bits
int	entier	32
long	entier long	64
float	nombre à virgule flottante	32 bits
double	nombre a virgule flottante	64 bits
char	caractère simple	16 bit
boolean	valeur booléenne	'true' ou 'false'

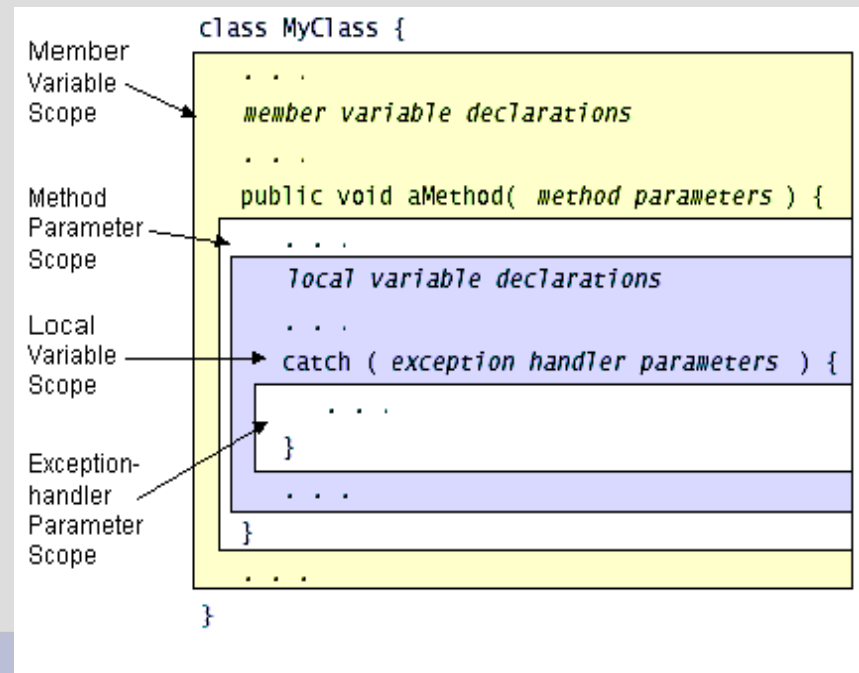
Langage Java

- Format des données : variables
 - Noms des variables
 - Identifiant (suite de caractères Unicode commençant par une lettre)
 - Pas un mot clé, un booléen ou un 'null'
 - Liste des mots clé

abstract	double	int	strictfp
boolean	else	interface	super
break	extends	long	switch
byte	final	native	synchronized
case	finally	new	this
catch	float	package	throw
char	for	private	throws
class	goto	protected	transient
const	if	public	try
continue	implements	return	void
default	import	short	volatile
do	instanceof	static	while

Langage Java

- Format des données : variables
 - Domaine de définition
 - C'est la région du programme d'où une variable peut être appelée par son nom.





Langage Java

- Format des données : variables
 - Variables finales
 - Une variable finale est une variable dont la valeur ne peut pas être modifiée.
 - Exemple : `final int afinalVar = 0;`



Le langage Java

- Formats de données
 - Concepts Objets
 - Variables
 - **Opérateurs**
 - Expressions et spécifications
 - Contrôle de flux



Langage Java

- Format des données : opérateurs
 - Ils sont de trois types :
 - **Unaire** : un opérande (Ex : ++), notation préfixe ou postfixe,
 - **Binaire** : deux opérandes (Ex : =), notation infixe,
 - **Ternaire** : trois opérandes (Ex, if-else : ? :), notation infixe.
 - Ils évaluent les expressions, et retournent une valeur

Langage Java

- Format des données : opérateurs
 - Opérateurs arithmétiques binaires

Opérateur	Usage	Description
+	op1 + op2	ajoute op1 et op2
-	op1 - op2	soustrait op2 de op1
*	op1 * op2	multiplie op1 et op2
/	op1 / op2	divise op1 et op2
%	op1 % op2	reste de la division de op1 par op2

Langage Java

- Format des données : opérateurs
 - Opérateurs arithmétiques unaires

Opérateur	Usage	Description
-	-op	Opposé de op
++	op++	op = op + 1 ; retourne op avant l'incrément
++	++op	op = op + 1 ; retourne op après l'incrément
--	op--	op = op - 1 ; retourne op avant l'incrément
--	--op	op = op - 1 ; retourne op après l'incrément

Langage Java

- Format des données : opérateurs
 - Opérateurs relationnels

Opérateur	Usage	Description
>	op1 > op2	op1 plus grand que op2
>=	op1 >= op2	op1 plus grand que ou égal à op2
<	op1 < op2	op1 plus petit que op2
<=	op1 <= op2	op1 plus petit que ou égal à op2
==	op1 == op2	op1 égal à op2
!=	op1 != op2	op1 différent de op2

Langage Java

- Format des données : opérateurs
 - Opérateurs conditionnels

Opérateurs	Usage	Vrai si
&&	op1 && op2	op1 et op2 vrais
	op1 op2	op1 ou op2 vrai
!	! op	op faux
&	op1 & op2	op1 et op2 vrais
	op1 op2	op1 ou op2 vrai
^	op1 ^ op2	op1 différent de op2

Langage Java

- Format des données : opérateurs
 - Opérateurs de décalage (shift)

Opérateur	Usage	Opération
>>	op1 >> op2	décale op1 de op2 bits vers la droite
<<	op1 << op2	décale op1 de op2 bits vers la gauche
>>>	op1 >>> op2	décale op1 de op2 bits vers la droite (non signé)

Langage Java

- Format des données : opérateurs
 - Opérateurs logiques

Opérateur	Usage	Opération
&	op1 & op2	and bit à bit
	op1 op2	or bit à bit
^	op1 ^ op2	xor bit à bit
~	~op	complément à deux bit à bit

Langage Java

- Format des données : opérateurs
 - Opérateurs d'assignement

Opérateur	Usage	Equivalent
<code>+=</code>	<code>op1 += op2</code>	<code>op1 = op1 + op2</code>
<code>-=</code>	<code>op1 -= op2</code>	<code>op1 = op1 - op2</code>
<code>*=</code>	<code>op1 *= op2</code>	<code>op1 = op1 * op2</code>
<code>/=</code>	<code>op1 /= op2</code>	<code>op1 = op1 / op2</code>
<code>%=</code>	<code>op1 %= op2</code>	<code>op1 = op1 % op2</code>
<code>&=</code>	<code>op1 &= op2</code>	<code>op1 = op1 & op2</code>
<code> =</code>	<code>op1 = op2</code>	<code>op1 = op1 op2</code>
<code>^=</code>	<code>op1 ^= op2</code>	<code>op1 = op1 ^ op2</code>
<code><<=</code>	<code>op1 <<= op2</code>	<code>op1 = op1 <<= op2</code>
<code>>>=</code>	<code>op1 >>= op2</code>	<code>op1 = op1 >>= op2</code>
<code>>>>=</code>	<code>op1 >>>= op2</code>	<code>op1 = op1 >>>= op2</code>

Langage Java

- Format des données : opérateurs
 - Autres opérateurs

Opérateur	Description
? :	if-else
[]	déclaration, création, accès aux 'arrays'
.	pour former des noms qualifiés
(params)	délimite une liste de paramètres
(type)	convertit une variable dans un type donné
new	crée un nouvel objet
op1 instanceof op2	détermine si op1 est une instance de op2



Le langage Java

- Formats de données
 - Concepts Objets
 - Variables
 - Opérateurs
 - **Expressions et spécifications**
 - Contrôle de flux



Langage Java

- Expression et spécifications
 - **Expression** : série de variables, opérateurs et appels de méthodes qui retournent une seule valeur.
 - **Spécifications** (Statement) : déclarations de variable, expressions terminées par un point virgule (;), ou bloc (if, for, while).
 - **Bloc** : ensemble de spécifications entre accolades ({}).



Langage Java

- Expression et spécifications
 - Les opérateurs d'assignement sont évalués de droite à gauche
 - Les autres opérateurs sont évalués de gauche à droite



Le langage Java

- Formats de données
 - Concepts Objets
 - Variables
 - Opérateurs
 - Expressions et spécifications
 - **Contrôle de flux**



Langage Java

- Commandes de contrôle de flux
 - Boucles
 - while(expression) {statement},
 - do {statement} while(expression),
 - for(initialisation ;fin ;modification de la condition) {statement},
 - Prise de décision
 - if(expression) {statement} else{statement},
 - switch-case,



Langage Java

- Commandes de contrôle de flux
 - Gestion d'exceptions
 - try-catch-finally,
 - throw,
 - Branchements
 - break,
 - continue,
 - label :,
 - return.

Langage Java

- Commandes de contrôle de flux
 - La spécification switch
 - effectuer différents blocs de commande en fonction de la valeur d'une variable
 - Eviter les 'forêts d'if'
 - Exemple :

```
public class SwitchDemo {  
    public static void main(String[] args) {  
  
        int month = 8;  
        switch (month) {  
            case 1: System.out.println("January"); break;  
            case 2: System.out.println("February"); break;  
            case 3: System.out.println("March"); break;  
            case 4: System.out.println("April"); break;  
            case 5: System.out.println("May"); break;  
            case 6: System.out.println("June"); break;  
            case 7: System.out.println("July"); break;  
            case 8: System.out.println("August"); break;  
            case 9: System.out.println("September"); break;  
            case 10: System.out.println("October"); break;  
  
            case 11: System.out.println("November"); break;  
            case 12: System.out.println("December"); break;  
        }  
    }  
}
```



- 'break'

```
public class SwitchDemo2 {
    public static void main(String[] args) {

        int month = 2;
        int year = 2000;
        int numDays = 0;

        switch (month) {
            case 1:
            case 3:
            case 5:
            case 7:
            case 8:
            case 10:
            case 12:
                numDays = 31;
                break;
            case 4:
            case 6:
            case 9:
            case 11:
                numDays = 30;
                break;
            case 2:
                if ( ((year % 4 == 0) && !(year % 100 == 0))
                    || (year % 400 == 0) )
                    numDays = 29;
                else
                    numDays = 28;
                break;
        }
        System.out.println("Number of Days = " + numDays);
    }
}
```

Langage Java

- Commandes de contrôle de flux
 - Spécification try-catch-finally
 - Try : une exception est susceptible d'être émise
 - Si une exception est émise dans try, les spécifications ne sont pas évaluées
 - Catch : Gestion de l'exception
 - Finally : spécifications effectuées indépendamment de la survenue d'une exception.

```
try {  
    statement(s)  
}  
catch (exceptiontype name) {  
    statement(s)  
}  
finally {  
    statement(s)  
}
```



Le langage Java

- Sommaire
 - Formats de données
 - **Premiers Objets**
 - Exceptions
 - Structure de programme
 - Input / Output



Le langage Java

- Premiers Objets
 - **Cycle de vie**
 - Caractères
 - Nombres
 - Tableaux



Langage Java

- Premiers Objets : Cycle de vie
 - Création
 - Interaction par messages entre les objets
 - Garbage collection
 - Destruction, et allocation des ressources à d'autres objets



Langage Java

- Premiers Objets : Cycle de vie - création
 - Exemple
 - Déclaration, en rouge, association du nom avec un type
 - Instantiation, par 'new', allocation d'espace mémoire à l'objet
 - Initialisation, par appel au constructeur, qui initialise l'objet

```
Point origin_one = new Point(23, 94);  
Rectangle rect_one = new Rectangle(origin_one, 100, 200);  
Rectangle rect_two = new Rectangle(50, 100);
```



Langage Java

- Premiers Objets : Cycle de vie - création
 - Constructeurs (1)
 - Pas de type de retour
 - Plusieurs possibles, avec des paramètres différents
 - Par défaut, sans paramètres

```
public class Point {  
    public int x = 0;  
    public int y = 0;  
    // a constructor!  
    public Point(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
}
```



Langage Java

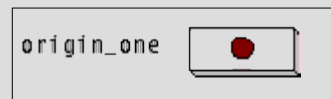
- Premiers Objets : Cycle de vie - création
 - Constructeurs (2)

```
public class Rectangle {  
    public int width = 0;  
    public int height = 0;  
    public Point origin;  
  
    // four constructors  
    public Rectangle() {  
        origin = new Point(0, 0);  
    }  
    public Rectangle(Point p) {  
        origin = p;  
    }  
    //...
```

```
//...  
    public Rectangle(int w, int h) {  
        this(new Point(0, 0), w, h);  
    }  
    public Rectangle(Point p, int w,  
int h) {  
        origin = p;  
        width = w;  
        height = h;  
    }  
    // methodes ...  
}
```

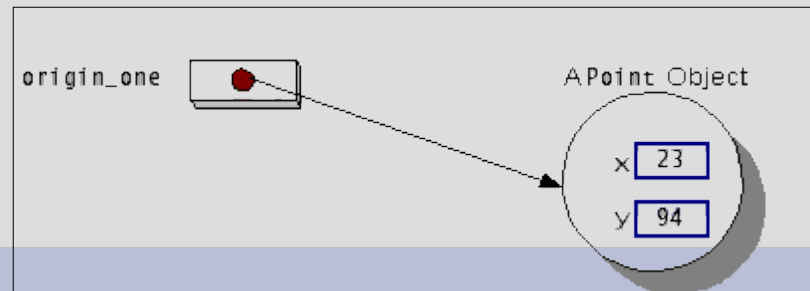
Langage Java

- Premiers Objets : Cycle de vie - création
 - Déclaration
 - Type nom
 - Utilisation d'une variable appelée 'nom', de type 'Type'
 - Les types sont des types élémentaires (int, etc...), des Classes ou des Interfaces
 - Pas de création d'objet, mais simple référence (pointeur)
 - Pour créer un objet, il faut l'instancier



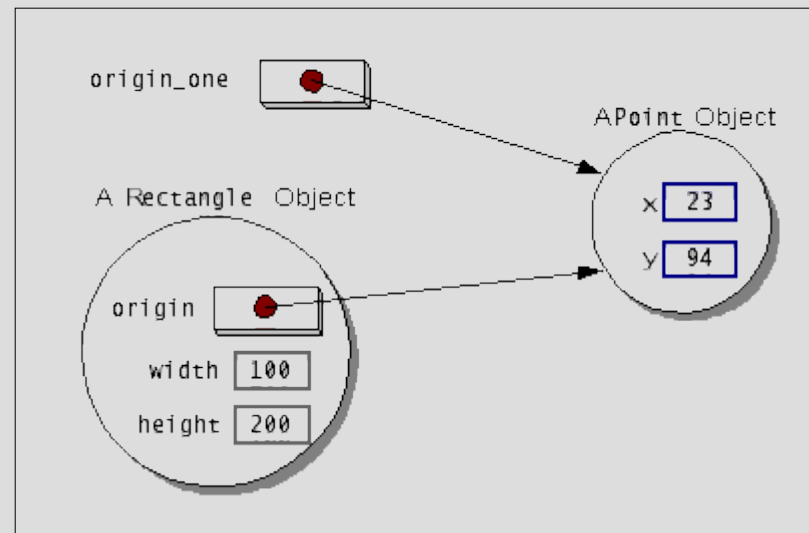
Langage Java

- Premiers Objets : Cycle de vie - création
 - Instanciación
 - Opérateur unaire new
 - Suivi du nom du constructeur (= nom de la Classe à instancier)
 - Type de retour : Objet
 - Assignation de la référence sur l'objet à une variable
 - Sinon, l'Objet est inatteignable
 - Initialisation par le constructeur



Langage Java

- Premiers Objets : Cycle de vie - création
 - Instanciation - exemple
 - Plusieurs références possibles sur un même Objet
 - `Rectangle rect_one = new Rectangle(origin_one, 100, 200);`





Langage Java

- Premiers Objets : Cycle de vie – Interactions
 - Types d'interactions entre objets :
 - Récupération d'informations
 - Modification d'état
 - Réalisation d'action(s)
 - Moyens :
 - Manipulation des variables
 - Appels de méthodes



Langage Java

- Premiers Objets : Cycle de vie – Interactions
 - Manipulations de variables
 - Nom qualifié : `objectReference.variableName`
 - Exemple : `Byte.MAX_VALUE`;
 - Nom simple, si on est dans l'espace de visibilité de la variable :
`variableName`
 - Valeurs par défaut, si elles existent :
 - `int height = new Rectangle().height`;
 - A noter : pas de référence sur le rectangle, donc l'Objet est détruit immédiatement



Langage Java

- Premiers Objets : Cycle de vie – Interactions
 - Accès aux variables
 - Déconseillé depuis d'autres classes
 - Risque d'incohérences (ex : valeurs négatives des cotés du rectangle)
 - Getter / setter :
 - Méthodes d'accès (`getValeur()`)
 - Méthodes de manipulation (`setValeur(int valeur)`)
 - Si accessibles depuis l'extérieur, partie de l'API
 - Pas de modification de nom ou de type possible



Langage Java

- Premiers Objets : Cycle de vie – Interactions
 - Appels de méthodes
 - Noms qualifiés :
 - `objectReference.methodName();`
 - `objectReference.methodName(argumentList);`



Langage Java

- Premiers Objets : Cycle de vie – Interactions
 - Exemple

```
public class Rectangle {  
    //attributs et constructeurs  
  
    // a method for moving the rectangle  
    public void move(int x, int y) {  
        origin.x = x;  
        origin.y = y;  
    }  
  
    // a method for computing the area of the rectangle  
    public int area() {  
        return width * height;  
    }  
}
```



Langage Java

- Premiers Objets : Cycle de vie – Interactions
 - Appels en cascade
 - `int areaOfRectangle = new Rectangle(100, 50).area();`



Langage Java

- Premiers Objets : Cycle de vie – Garbage Collection
 - Gestion implicite de la mémoire
 - Évite perte de temps et sources d'erreurs
 - Destruction des objets inutilisés par la plate-forme
 - Garbage Collection
 - Quand plus de référence sur un objet
 - Variable hors du champ de validité
 - `Variable := null;`



Langage Java

- Premiers Objets : Cycle de vie – Garbage Collection
 - Destruction automatique des objets non référencés
 - Méthode `System.gc()` pour l'appel explicite
 - Après du code utilisant beaucoup de mémoire, et avant un autre code nécessitant beaucoup de mémoire
 - Méthode `finalize()`
 - Pour finaliser un objet
 - Certains objets ne sont pas gérés par le Garbage Collector (ex : code natif)



Langage Java

- Premiers Objets : caractères
 - 3 Classes
 - Character : un caractère
 - String : une chaine de caractères non modifiable
 - StringBuffer : une chaine de caractères modifiable



Le langage Java

- Premiers Objets
 - Cycle de vie
 - **Caractères**
 - Nombres
 - Tableaux



Langage Java

- Premiers Objets : caractères
 - Character
 - `java.lang.Character`
 - `Character a1 = new Character('a');`
 - `int difference = a1.compareTo(b);`
 - `boolean a1.equals(a2)`
 - `String a.toString()`
 - `char a.charValue()`
 - `Character.isUpperCase(a.charValue())`



Langage Java

- Premiers Objets : caractères

- Les chaines

- String, statique, plus performant
- StringBuffer, construction dynamique de texte, de commandes, etc.

```
public class StringsDemo {  
    public static void main(String[] args) {  
        String palindrome = "Dot saw I was Tod";  
        int len = palindrome.length();  
        StringBuffer dest = new StringBuffer(len);  
  
        for (int i = (len - 1); i >= 0; i--) {  
            dest.append(palindrome.charAt(i));  
        }  
        System.out.println(dest.toString());  
    }  
}
```



Langage Java

- Premiers Objets : caractères
 - Les chaines - création

```
String palindrome = "Dot saw I was Tod";
```

```
char[] helloArray = { 'h', 'e', 'l', 'l', 'o' };  
String helloString = new String(helloArray);  
System.out.println(helloString);
```

```
String palindrome = "Dot saw I was Tod";  
int len = palindrome.length();  
StringBuffer dest = new StringBuffer(len);
```



Langage Java

- Premiers Objets : caractères
 - Les chaines – méthodes
 - Length()
 - charAt(int i) : caractère d'indice i (premier caractère, indice 0, dernier caractère, indice n-1)
 - substring(int i), substring(int i, int j) : sous-chaîne commençant à i, finissant à j si spécifié, à la fin sinon



Langage Java

- Premiers Objets : caractères
 - String – méthodes
 - indexOf(recherche) : indice de la première apparition de 'recherche'
 - lastIndexOf(recherche) : indice de la dernière apparition de 'recherche'
 - StringBuffer - méthodes
 - capacity(), espace mémoire alloué



Langage Java

- Premiers Objets : caractères
 - StringBuffer – modifications
 - append(type), pour des types variés (char, String, float, int, boolean, Object)
 - toString()
 - insert(int i, String texte), insert un texte donné à l'indice i
 - setCharAt, remplacement d'un caractère d'indice donné



Langage Java

- Premiers Objets : caractères
 - Conversion en String
 - `String StringBuffer.toString();`
 - `String String.valueOf(Object);`
 - Conversion de String en nombre
 - `Float.valueOf(String);`



Langage Java

- Premiers Objets : caractères
 - Concaténation
 - Pas possible sur les String théoriquement
 - Mais conversion par le compiler
 - Opérateur +
 - Exemples :

```
String cat = "cat";  
System.out.println("con" + cat + "enation");  
equivaut à :  
String cat = "cat";  
System.out.println(new StringBuffer().append("con").  
                                append(cat).append("enation").toString());
```

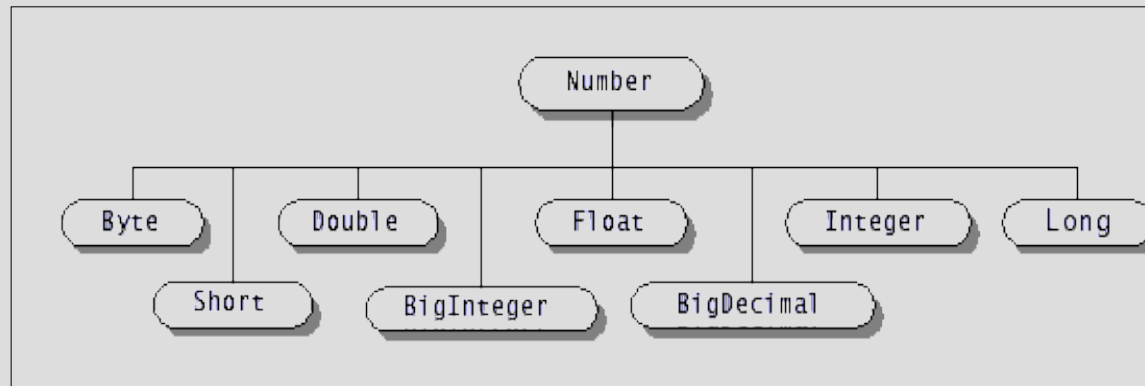


Le langage Java

- Premiers Objets
 - Cycle de vie
 - Caractères
 - **Nombres**
 - Tableaux

Langage Java

- Premiers Objets : nombres
 - Number



- NumberFormat
- DecimalFormat



Langage Java

- Premiers Objets : nombres
 - Types primitifs
 - double
 - int
 - ...
 - Types de wrapper
 - Double
 - Integer
 - ...



Langage Java

- Premiers Objets : nombres
 - Java.text.NumberFormat - exemple

```
Double amount = new Double(345987.246);  
NumberFormat numberFormatter;  
String amountOut;  
  
numberFormatter = NumberFormat.getNumberInstance(currentLocale);  
amountOut = numberFormatter.format(amount);  
System.out.println(amountOut + " " + currentLocale.toString());
```

345 987,246	fr_FR
345.987,246	de_DE
345,987.246	en_US



Langage Java

- Premiers Objets : nombres
 - Monnaie - exemple

```
Double currency = new Double(9876543.21);  
NumberFormat currencyFormatter;  
String currencyOut;  
  
currencyFormatter = NumberFormat.getCurrencyInstance(currentLo-  
cale);  
currencyOut = currencyFormatter.format(currency);  
System.out.println(currencyOut + " " + currentLocale.toString());
```

9 876 543,21 F	fr_FR
9.876.543,21 DM	de_DE
\$9,876,543.21	en_US



Langage Java

- Premiers Objets : nombres
 - Pourcentage - exemple

```
Double percent = new Double(0.75);  
NumberFormat percentFormatter;  
String percentOut;
```

```
percentFormatter = NumberFormat.getPercentInstance(currentLocale);  
percentOut = percentFormatter.format(percent);
```

Langage Java

- Premiers Objets : nombres
 - `java.text.DecimalFormat` : exemple

```
DecimalFormat myFormatter = new DecimalFormat(pattern);  
String output = myFormatter.format(value);  
System.out.println(value + " " + pattern + " " + output);
```

valeur	pattern	output
123456.789	###.###.###	123,456.789
123456.789	###.##	123456.79
123.78	000000.000	000123.780
12345.67	\$###.###.###	\$12,345.67
12345.67	###.###.###	12,345.67



Langage Java

- Premiers Objets : nombres
 - `java.text.DecimalFormatSymbols` : exemple

```
DecimalFormatSymbols unusualSymbols =  
    new DecimalFormatSymbols(currentLocale);  
unusualSymbols.setDecimalSeparator('|');  
unusualSymbols.setGroupingSeparator('^');  
  
String strange = "#,##0.###";  
DecimalFormat weirdFormatter =  
    new DecimalFormat(strange, unusualSymbols);  
weirdFormatter.setGroupingSize(4);  
  
String bizarre = weirdFormatter.format(12345.678);  
System.out.println(bizarre);
```



Langage Java

- Premiers Objets : nombres
 - DecimalFormatsymbols : exemple
 - Output : 1^2345|678
 - Et aussi
 - Number.MIN_VALUE
 - Number.MAX_VALUE
 - Classe Math
 - fonction mathématique, génération de nombres aléatoires

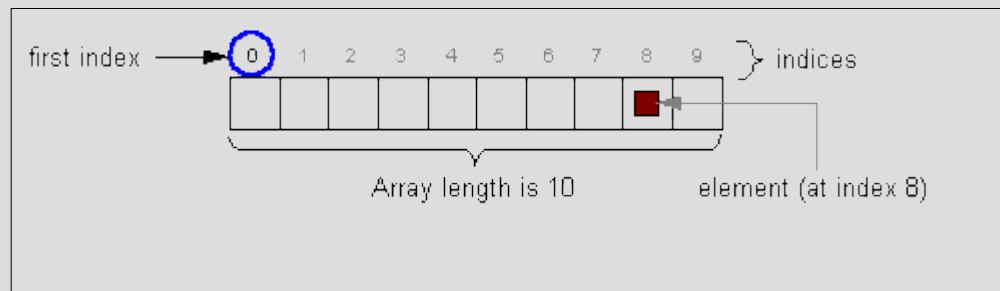


Le langage Java

- Premiers Objets
 - Cycle de vie
 - Caractères
 - Nombres
 - **Tableaux**

Langage Java

- Premiers Objets : tableaux
 - Array : structure de longueur fixe contenant des objets de même type



- Structure de données variables : Collection, Vector



Langage Java

- Premiers Objets : tableaux
 - Opérations sur les tableaux :
 - Déclaration
 - Création
 - Accession
 - Connaître la taille
 - Initialisation



Langage Java

- Premiers Objets : tableaux
 - Exemple

```
public class ArrayDemo {  
    public static void main(String[] args) {  
        int[] anArray;           // declare an array of integers  
  
        anArray = new int[10];   // create an array of integers  
  
        // assign a value to each array element and print  
        for (int i = 0; i < anArray.length; i++) {  
            anArray[i] = i;  
            System.out.print(anArray[i] + " ");  
        }  
        System.out.println();  
    }  
}
```



Langage Java

- Premiers Objets : tableaux
 - Déclaration

```
int[] anArray;  
float[] anArrayOfFloats;  
boolean[] anArrayOfBooleans;  
Object[] anArrayOfObjects;  
String[] anArrayOfStrings;
```

- Création

```
new elementType[arraySize];
```



Langage Java

- Premiers Objets : tableaux

- Accession

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

- Connaître la taille

```
arrayname.length
```



Langage Java

- Premiers Objets : tableaux

- Initialisation

- Noter : la taille de l'Array dépend du nombre de valeurs indiquées

```
boolean[] answers = { true, false, true, true, false };
```

- Tableaux d'Objets

```
public class ArrayOfStringsDemo {  
    public static void main(String[] args) {  
        String[] anArray = { "String One", "String Two", "String  
Three" };  
  
        for (int i = 0; i < anArray.length; i++) {  
            System.out.println(anArray[i].toLowerCase());  
        }  
    }  
}
```



Langage Java

- Premiers Objets : tableaux
 - Tableaux de tableaux

```
public class ArrayOfArraysDemo {  
    public static void main(String[] args) {  
        String[][] cartoons =  
        {  
            { "Flintstones", "Fred", "Wilma", "Pebbles", "Dino" },  
            { "Rubbles", "Barney", "Betty", "Bam Bam" },  
            { "Jetsons", "George", "Jane", "Elroy", "Judy",  
"Rosie", "Astro" },  
            { "Scooby Doo Gang", "Scooby Doo", "Shaggy", "Velma",  
"Fred", "Daphne" }  
        };  
        // ...  
    }  
}
```



Langage Java

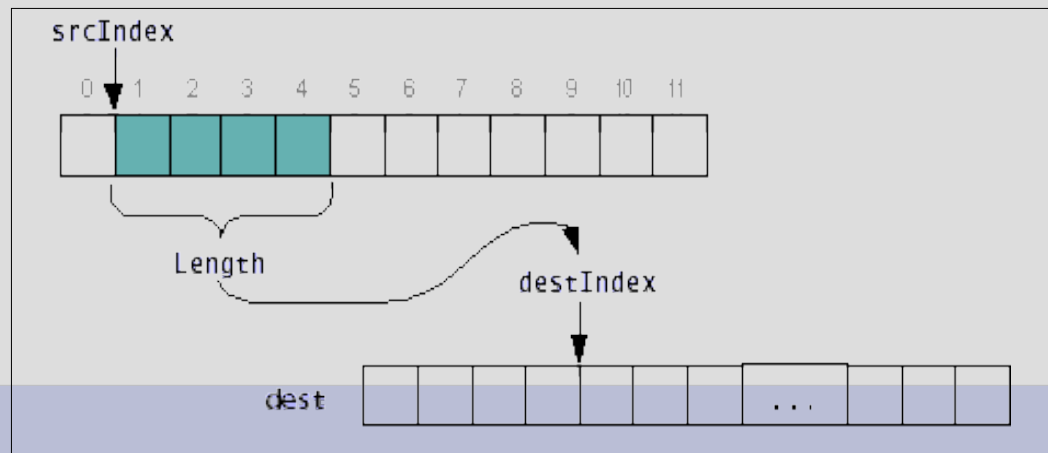
- Premiers Objets : tableaux
 - Tableaux de tableaux

```
//...  
for (int i = 0; i < cartoons.length; i++) {  
    System.out.print(cartoons[i][0] + ": ");  
    for (int j = 1; j < cartoons[i].length; j++) {  
        System.out.print(cartoons[i][j] + " ");  
    }  
    System.out.println();  
}  
}
```

Langage Java

- Premiers Objets : tableaux
 - Copie de tableaux
 - System.arraycopy()

```
public static  
    void arraycopy(Object source,  
                   int srcIndex,  
                   Object dest,  
                   int destIndex,  
                   int length)
```





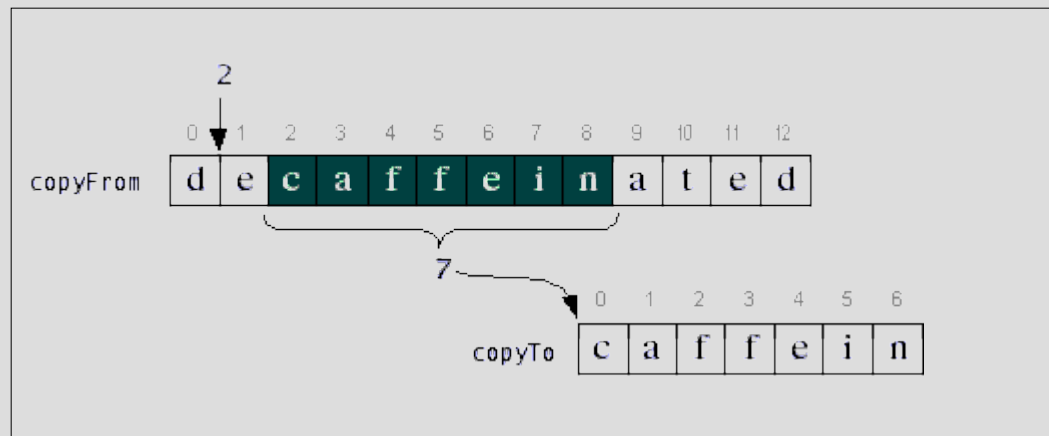
Langage Java

- Premiers Objets : tableaux
 - Copie de tableaux

```
public class ArrayCopyDemo {  
    public static void main(String[] args) {  
        char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e',  
                             'i', 'n', 'a', 't', 'e', 'd' };  
        char[] copyTo = new char[7];  
  
        System.arraycopy(copyFrom, 2, copyTo, 0, 7);  
        System.out.println(new String(copyTo));  
    }  
}
```

Langage Java

- Premiers Objets : tableaux
 - Copie de tableaux
 - Attention à ce que la longueur du tableau de destination soit suffisante





Le langage Java

- Sommaire
 - Formats de données
 - Premiers Objets
 - **Exceptions**
 - Structure de programme
 - Input / Output



Langage Java

- Exceptions

'Evenement qui apparait pendant le déroulement d'un programme et perturbe le flux d'instructions'



Langage Java

- Exceptions
 - Gestion des erreurs
 - Séparée du corps du programme
 - Confinement
 - Et propagation dans la pile d'appel ayant mené à l'erreur
 - Groupement et différenciation
 - Des erreurs
 - Permet un traitement adapté
 - Stabilité du programme



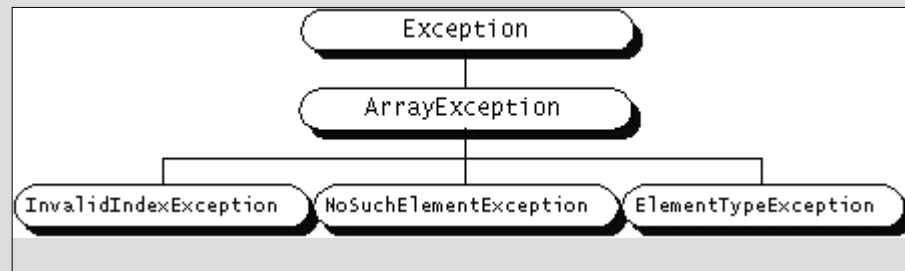
Langage Java

- Exceptions
 - Exemple : Propagation des erreurs dans la pile d'appel

```
method1 {  
    call method2;  
}  
method2 {  
    call method3;  
}  
method3 {  
    call readFile;  
}
```

Langage Java

- Exceptions
 - Exemple : Groupement et différenciation





Langage Java

- Exceptions
 - Peuvent être émise par
 - Des méthodes des API utilisées
 - Des méthodes de notre programme
 - Doivent être interceptées

```
Try{ /*code pouvant être erroné*/  
}  
catch(MonException e){  
    /*gestion de l'exception*/  
}  
finally{ /*finalisation*/  
}
```



Langage Java

- Exceptions
 - Exemple

```
public void essaie(){  
    System.out.println("Catching, 1");  
    try{  
        System.out.println("Catching, 2");  
        Relais r = new Relais();  
        r.relaie();  
        System.out.println("Catching, 3");  
    }  
    catch(LanceException le){  
        System.out.println("Exception Lance interceptée");  
    }  
}
```



Langage Java

- Exceptions
 - Gestion d'exceptions multiples
 - `catch(Exception e)`, générique
 - Différentiation de traitement - exemple

```
...  
} catch (monException m) {  
    ...  
} catch (Exception e) {  
    ...  
}
```



Langage Java

- Exceptions
 - Exception lors d'appel de méthodes
 - Peuvent être interceptées
 - Peuvent être relayées : mot clé throws
 - Nombre de relais indéterminé
 - Traitement dans une autre classe



Langage Java

- Exceptions
 - Exception lors d'appel de méthodes
 - Exemple :

```
public void relaie() throws LanceException{  
  
    System.out.println("Relais, 1");  
    Lanceur armstrong = new Lanceur();  
    System.out.println("Relais, 2");  
    armstrong.lance();  
    System.out.println("Relais, 3");  
}
```



Langage Java

- Exceptions
 - Emission d'exception
 - Si erreur
 - Mot clé throw
 - Lancement d'un exception existante
 - Ou lancement d'une exception spécifique



Langage Java

- Exceptions
 - Emission d'exception
 - Exemple :

```
public void lance() throws LanceException{

    System.out.println("Lanceur, 1");
    boolean echoue = test();//si test faux, echoue = false
    if(echoue) {
        System.out.println("echec");
        System.out.println("Lanceur, 2");
        throw new LanceException();
        // unreachable : System.out.println("Lanceur, 3");
    }
}
```



Langage Java

- Exceptions
 - Execution du code
 - Le code est exécuté jusqu'à la méthode qui transmet l'exception
 - Les spécifications suivantes ne sont pas accessibles
 - Si lancement d'une exception (throw), le code qui suit est non valide
 - Unreachable
 - Erreur lors de la compilation



Langage Java

- Exceptions
 - Execution du code – Exemples
 - Transmission d'exception

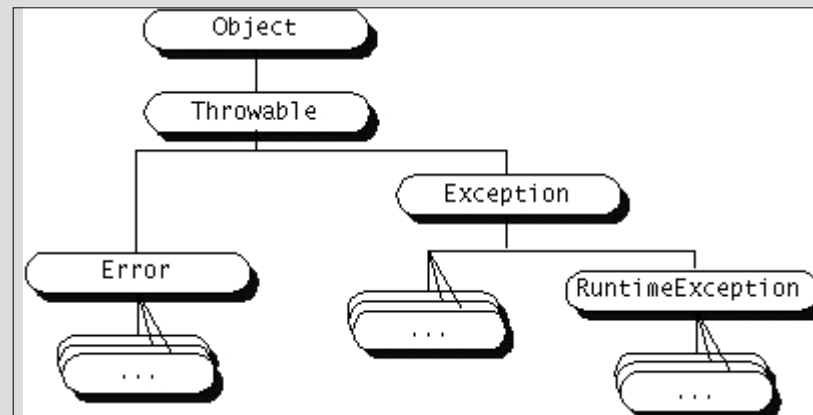
```
armstrong.lance();  
System.out.println("Relais, 3"); //not executed
```

- Emission d'une exception

```
throw new LanceException();  
// unreachable : System.out.println("Lanceur, 3");
```

Langage Java

- Exceptions
 - La classe MonException
 - Erreur non critique pour le système
 - Hérite de Throwable, ou Exception
 - Pas de contenu particulier nécessaire
 - Classe Error : erreur système





Langage Java

- Exceptions
 - La classe MonException
 - Exemple

```
package monPackage;  
  
public class LanceException extends Exception{  
  
    }  
}
```



Langage Java

- Exceptions
 - Exceptions existantes
 - Quelques exemples
 - NegativeArraySizeException – tentative de création d'un tableau avec une taille négative
 - RuntimeException – Exception dans la machine virtuelle pendant l'exécution
 - NullPointerException – accès à un objet par une référence '**null**'



Le langage Java

- Sommaire
 - Formats de données
 - Premiers Objets
 - Exceptions
 - **Structure de programme**
 - Input / Output



Le langage Java

- Structure de programme
 - **Héritage**
 - Interface
 - Classes abstraites
 - Inner Class
 - Threads

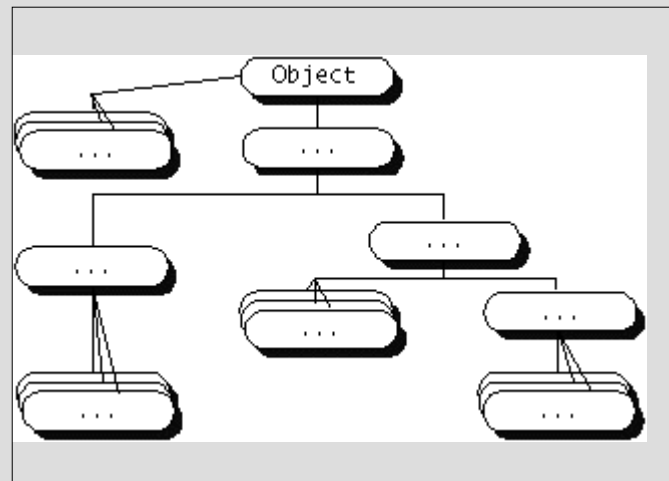


Langage Java

- Structure : Héritage
 - Appel 'extends'
 - Dérive une classe comme sous-classe d'une autre
 - Pas d'héritage multiple
 - Tout hérite de `java.lang.Object`

Langage Java

- Structure : Héritage
 - Object : comportement général
 - Bas de la hiérarchie : comportement spécifique
 - Intermédiaire : spécialisation progressive





Langage Java

- Structure : Héritage - Sous-classe
 - 'Classe qui étend une autre'
 - Hérite de l'état (variables) et du comportement (méthodes) de tous ses ancêtres
 - Ancêtres : classe mère, ainsi que tous les ascendants
 - Exemple :
 - `java.lang.Object`
 - `java.lang.Number`
 - `java.lang.Integer`



Langage Java

- Structure : Héritage
 - Membres des ses ancêtres
 - Membres : variable + méthodes
 - Attention : les constructeurs non membres, donc non hérités
 - On peut :
 - Cacher les variables
 - Écraser les méthodes en les réécrivant



Langage Java

- Structure : Héritage
 - Membres hérités :
 - Public ou protected
 - Sans spécification (public, protected, private), si la sous-classe et la classe mère sont dans le même package
 - Ceux qui ne sont pas écrasés dans la sous-classe



Langage Java

- Structure : Héritage

- Exemple :

```
class Super {  
    Number aNumber;  
}  
class Subbie extends Super {  
    Float aNumber;  
}
```

- Accès aux variables cachées de la superclasse
 - `super.aNumber`



Langage Java

- Structure : Héritage
 - Ecraser des méthodes
 - La nouvelle méthode doit avoir
 - le même nom
 - Le même type de retour
 - Les mêmes paramètres
 - Sinon, les méthodes de la classe mère et de la classe fille coexistent



Langage Java

- Structure : Héritage
 - Ecraser des méthodes
 - Exemple : `Object.toString` renvoie le nom de la classe et le Hash Code
 - `java.lang.Object@194df86`
 - On peut vouloir retourner quelque chose de plus utile
 - Modification du spécifieur
 - Protected -> public
 - Donner plus d'accès, pas moins



Langage Java

- Structure : Héritage
 - Appel d'une méthode écrasée
 - `super.overriddenMethodName();`
 - Appel du constructeur de la classe mère
 - Non membre
 - Non hérité
 - Code : `Public MaClasse(){super();}`
 - Méthodes finales
 - Ne peuvent pas être écrasées
 - `Final public String getName();`



Langage Java

- Structure : Héritage
 - La classe `java.lang.Object`
 - Contient des méthodes de :
 - Comparaison
 - Conversion en String
 - Attente d'une condition
 - Notification d'une condition
 - Indication de classe



Langage Java

- Structure : Héritage
 - La classe `java.lang.Object`
 - Méthodes modifiables
 - `clone`
 - `equals` / `hashCode`
 - `finalize`
 - `toString`
 - Méthodes finales
 - `getClass`
 - `notify`
 - `notifyAll`
 - `wait`



Langage Java

- Structure : Héritage
 - Classes 'final'
 - Sécurité
 - Éviter de remplacer une classe par une sous-classe malveillante
 - Exemple : String
 - Vérification dans le compilateur ET l'interpréteur
- OO-design
 - Classes qui, logiquement, n'héritent pas
 - Exemple : `final class MaClasseParfaite{}`



Langage Java

- Structure : Héritage
 - Méthodes 'final'
 - Implémentation spécifique
 - Méthode critique dans le déroulement du programme



Le langage Java

- Structure de programme
 - Héritage
 - **Interface**
 - Classes abstraites
 - Inner Class
 - Threads



Langage Java

- Structure : Interfaces
 - Qu'est ce qu'une interface ?

*Ensemble de
définitions de méthodes et de constantes
sans implémentation*



Langage Java

- Structure : Interfaces
 - Qu'est ce qu'une interface ?
 - Exemple :

```
public interface ServeurInterface{  
  
    // seulement declaration de constantes  
    final int portNumber = 425;  
  
    // pas de constructeur définissable  
    public boolean connection(int AdresseIPClient);  
    public boolean deconnection(int AdresseIPClient);  
  
}
```



Langage Java

- Structure : Interfaces
 - Qu'est ce qu'une interface ?
 - Comportement type
 - Définition d'un ensemble de méthodes
 - Pas implémentées
 - Implémentation par n'importe quelle classe
 - Toutes les méthodes définies par cette interface
 - Donc comportement prédéfini

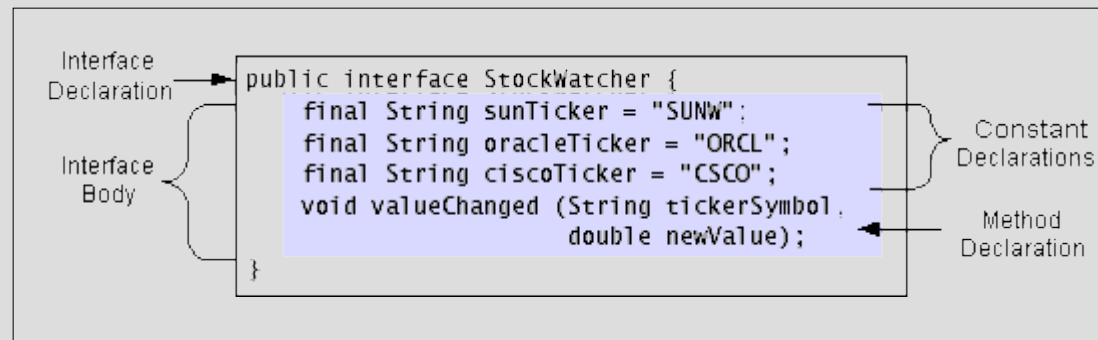


Langage Java

- Structure : Interfaces
 - Qu'est ce qu'une interface ?
 - Différence avec une classe abstraite
 - Aucune implémentation de méthodes
 - Une classe peut implémenter plusieurs interfaces
 - Mais n'a qu'une seule superclasse (abstraite ou non)
 - Une interface ne fait pas partie de la hierarchie des classes

Langage Java

- Structure : Interfaces
 - Définir une interface
 - Déclaration de l'interface
 - Corps de l'interface
 - Définition des constantes
 - Définitions des méthodes





Langage Java

- Structure : Interfaces
 - Définir une interface
 - Déclaration de l'interface
 - Si plusieurs super interfaces, séparées par des virgules

<code>public</code>	Makes this interface public.
<code>interface InterfaceName</code>	This is the name of the interface.
<code>Extends SuperInterfaces</code>	This interface's superinterfaces.
<pre>{ <i>InterfaceBody</i> }</pre>	



Langage Java

- Structure : Interfaces
 - Définir une interface
 - Définition des méthodes et constantes
 - Une définition de méthode est suivie par un ;
 - `void valueChanged(String tickerSymbol, double newValue);`
 - Méthodes public, abstract par défaut



Langage Java

- Structure : Interfaces
 - Implémenter une interface

```
public class Serveur implements ServeurInterface
```



Langage Java

- Structure : Interfaces
 - Implémenter une interface : Exemple

```
public class Serveur implements ServeurInterface{

    public Serveur(int IPServ){
        AdresseIPServeur = IPServ;
        System.out.println("création du serveur : "+ IPServ);}

    public boolean connection(int AdresseIPClient){
        ...
        return connect;
    }

    ...
}
```



Langage Java

- Structure : Interfaces
 - Implémenter une interface
 - Classe Interface : définition, mais pas d'usage directe
 - Idem classes abstraites
 - Classe d'implémentation : classe qui sera manipulée
 - Idem classes héritantes de classes abstraites
 - Différence : on peut implémenter plusieurs interfaces



Langage Java

- Structure : Interfaces
 - Implémenter une interface
 - Classe d'implémentation
 - Hérite des constantes
 - Doit
 - Soit implémenter toutes les méthodes
 - Soit être abstract



Langage Java

- Structure : Interfaces
 - Les interfaces types de données : Exemple

```
public class Client{  
  
    public static void openSession(ServeurInterface si){  
        si.connection(AdresseIPClient);  
    }  
  
    public static void main(String[] args){  
  
        Serveur web = new Serveur(1921683475);  
  
        // gestion locale de la connection  
        openSession(web);  
    }  
}
```



Langage Java

- Structure : Interfaces
 - Les interfaces types de données
 - Utilisées comme n'importe quel autre type de données
 - En paramètre de méthodes ou constructeur
 - Indique que seuls les objets implémentant cette interface peuvent être utilisés en paramètre



Langage Java

- Structure : Interfaces
 - Les interfaces ne peuvent pas grandir
 - Rajouter une méthode à une interface = modifier cette interface
 - Les Classes d'implémentation ne sont plus valides !
 - Elles n'implémentent plus toutes les méthodes de l'interface
 - Il faut créer une nouvelle interface qui hérite de la première
 - Exemple
 - Ajouter une méthode `getInformation()` à notre interface `ServeurInterface`



Langage Java

- Structure : Interfaces
 - Points communs Classe abstraite / Interface
 - Classe de définition non instanciable
 - Héritage des variables
 - Définition des méthodes
 - Les classes d'implémentation peuvent être étendues



Langage Java

- Structure : Interfaces
 - Différence Classe abstraite / Interface
 - On peut implémenter plusieurs interfaces
 - Classe abstraite
 - comportement défini, implémenté
 - Constructeur possible
 - Interface
 - comportement défini, sans implémentation
 - Pas de constructeur



Langage Java

- Structure : Interfaces
 - Conclusion
 - Interface = protocole de communication entre deux objets
 - Définition = déclaration + corps
 - Corps = constantes + déclaration de méthode
 - Pas d'implémentation
 - Classe d'implémentation : code pour toutes les méthodes



Le langage Java

- Structure de programme
 - Héritage
 - Interface
 - **Classes abstraites**
 - Inner Class
 - Threads



Langage Java

- Structure : Classes abstraites
 - Concepts abstraits
 - Non instanciées (pas d'utilisation directe)
 - Exemple :
 - nourriture, abstrait
 - Pain, chocolat, sandwich, concrets
- `Public abstract class Number`

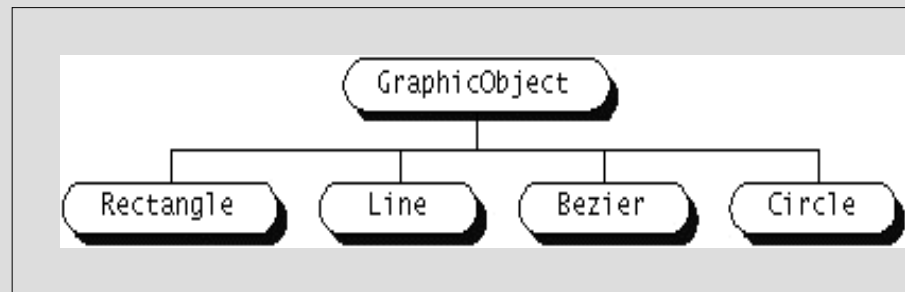


Langage Java

- Structure : Classes abstraites
 - Méthodes abstraites
 - Dans les classes abstraites
 - Sans implémentation
 - Pour la définition d'une interface de programmation

Langage Java

- Structure : Classes abstraites
 - Méthodes abstraites : exemple



- Classe d'objets graphiques
 - Abstract class GraphicObject
 - Class Rectangle, Line, Bezier, Circle



Langage Java

- Structure : Classes abstraites
 - Méthodes abstraites : exemple
 - Méthodes communes
 - moveTo(), etc.
 - Implémentation (code) dans la classe abstraite
- Méthodes spécifiques
 - Draw(), etc.
 - Implémentation dans les classes filles
 - Déclaration dans la classe abstraite



Langage Java

- Structure : Classes abstraites
 - Méthodes abstraites : exemple
 - Code de la classe abstraite

```
abstract class GraphicObject {  
    int x, y;  
    . . .  
    void moveTo(int newX, int newY) {  
        . . .  
    }  
    abstract void draw();  
}
```

Langage Java

- Structure : Classes abstraites
 - Méthodes abstraites : exemple
 - Code des classe filles

```
class Circle extends GraphicObject {  
    void draw() {  
        . . .  
    }  
}
```

```
class Rectangle extends GraphicObject {  
    void draw() {  
        . . .  
    }  
}
```



Langage Java

- Structure : Classes abstraites
 - Bilan
 - Classes filles : implémentation obligatoire des méthodes abstraites
 - Classe abstraite : avec ou sans méthodes abstraites
 - Méthode abstraite : dans une classe abstraite



Le langage Java

- Structure de programme
 - Héritage
 - Interface
 - Classes abstraites
 - **Inner Class**
 - Threads



Langage Java

- Structure : inner Classes



Le langage Java

- Structure de programme
 - Héritage
 - Interface
 - Classes abstraites
 - Inner Class
 - **Threads**



Langage Java

- Structure : Threads



Le langage Java

- Sommaire
 - Formats de données
 - Premiers Objets
 - Exceptions
 - Structure de programme
 - **Input / Output**



Langage Java

- Input / Output



Langage Java

- Bilan

