



Le langage Java

Classes de base et méthodes de base

Objectifs Pédagogiques

À l'issue de cette formation, vous serez en mesure de :

- ✓ Utiliser les classes de base de Java
- ✓ Savoir redéfinir la méthode equals

Programme détaillé ou sommaire

Les wrappers

La classe String

Les classes StringBuffer et StringBuilder

Les dates en Java 7 et inférieur

Redéfinition de equals

La javadoc

Chapitre 1

Les wrappers

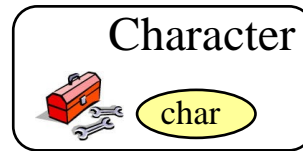
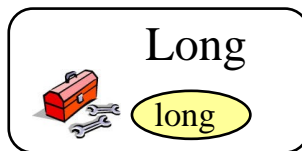
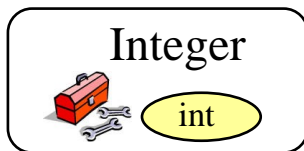
Les Wrappers (1/3)

Les types primitifs permettent d'effectuer des opérations simples et rapides

```
int i=0; int j=0;  
i = 2 + 3; j = i + 2;  
j++;
```

Pour chaque type primitif il existe une classe dite **Wrapper** (emballage).

Une classe Wrapper contient une variable d'un type primitif donné et des outils facilitant la manipulation de cette variable.



...

Les Wrappers (2/3)

Classes Wrapper existantes : Boolean, Byte, Short, Integer, Long, Float, Double, Character

Méthodes proposées par les classes Wrapper :

toString() conversion en chaîne de caractères

intValue(), **doubleValue()**, **longValue()** ... pour des conversions numériques

Les Wrappers de types numériques contiennent des constantes **MAX_VALUE** et **MIN_VALUE**

Les Wrappers (3/3)

Exemples d'utilisation

Méthodes d'instance

```
String s = i.toString();  
double d = i.doubleValue();  
boolean b2 = b.booleanValue();
```

Méthodes statiques

```
String s = "12";  
int i = Integer.parseInt(s);  
Integer i = Integer.valueOf(12);  
boolean b = Boolean.getBoolean("true");  
Boolean b = Boolean.valueOf(true);
```

new deprecated Java 9

~~Integer i = new Integer(12)~~
~~Boolean b = new Boolean(true);~~



la plupart de ces méthodes ne s'appliquent pas au type
Character qui n'enveloppe pas un nombre.

Chapitre 2

La classe String

String

Les String représentent les chaînes de caractères en Java

Les Strings sont des objets

Initialisation facilitée sans le mot-clé 'new'

La taille d'une String n'est pas limitée

```
String s1 = new String("hello world");  
String s2 = "hello world";
```

Deux modes d'initialisation d'une String



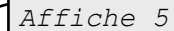
L'initialisation avec l'opérateur " exploite le String pool.

String – Méthode length()

Longueur de la chaîne

int length()

```
String s1 = new String("hello");  
int longueurChaine = s1.length();  
System.out.println(longueurChaine);
```



Affiche 5

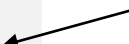
String – Méthode compareTo()

Comparaison de 2 chaînes de caractères

int compareTo(String)

```
String s1 = new String("hello");  
String s2 = new String("hello");  
System.out.println(s1.compareTo(s2));
```

Affiche 0




String – Méthode trim()

Manipulation de chaînes

`trim()` : supprime les espaces superflus en début et fin de chaîne

```
String s1 = new String("    hello world ");  
System.out.println(s1);  
System.out.println(s1.trim());
```



Affiche :
hello world
hello world

String – toLowerCase() et toUpperCase()

Mettre en majuscules, minuscules

toUpperCase(), toLowerCase() ...

```
String s1 = new String("hello world");  
System.out.println(s1.toUpperCase());
```



Affiche « HELLO WORLD »

String - Concaténation

Concaténation : concat(...) ou opérateur +

```
String s1 = new String("hello");  
String s2 = s1 + " world";  
String s3 = s1.concat(" world");
```



**Utilisée à grande échelle, la concaténation de String avec +
est une opération peu performante**

String - Immutabilité

Les instances de String sont dites *immutables*



```
String s2 = "hello world";  
s2 = "bye bye world";
```

L'objet référencé par s2 n'est plus le même

```
String s2 = " hello world ";  
s2.trim();
```



```
String s2 = " hello world ";  
s2 = s2.trim();
```



String – Méthode format (1/4)

*Méthode **static** qui permet de formater une chaîne de caractères en évitant l'utilisation de la concaténation :*

```
String str1 = String.format("Je suis %s %s et j'ai %f € en poche", "Lee", "Tim", 2.5);
```

Je suis Lee Tim et j'ai 2,500000 € en poche

String – Méthode format %s (2/4)

Le paramètre %s permet d'indiquer la position d'une String:

```
String str1 = String.format("Je suis %s %s et j'ai %f € en poche", "Lee", "Tim", 2.5);
```

Je suis Lee Tim et j'ai 2,500000 € en poche

Paramètre	Type
%s	Chaine de caractères
%d	Nombre entier
%f	Nombre décimal
%n	Permet d'ajouter un retour à la ligne

String – Méthode format %f (3/4)

Le %f est réservé aux nombres décimaux:

```
String str1 = String.format("J'ai %f€ en poche", 2.5);
```

J'ai 2,500000 € en poche

```
String str1 = String.format("J'ai %.1f€ en poche", 2.5);
```

J'ai 2,5 € en poche

```
String str1 = String.format("J'ai %.2f€ en poche", 2.5);
```

J'ai 2,50 € en poche

Ajout d'un padding à gauche:

```
String str1 = String.format("J'ai %5.2f€ en poche", 2.5);
```

J'ai 2,50 € en poche

2.5 est formaté sur 5 caractères et il doit y avoir 2 chiffres après la virgule.

	2	.	5	0
--	---	---	---	---

String – Méthode format %f et Locale (4/4)

La locale permet de fournir des paramètres culturels (langue, format des dates, etc.):

```
String str1 = String.format(Locale.US, "J'ai %5.2f € en poche", 2.5);
```

J'ai 2.50 € en poche

```
String str1 = String.format(Locale.FRANCE, "J'ai %5.2f € en poche", 2.5);
```

J'ai 2,50 € en poche

Chapitre 3

La classe StringBuilder

StringBuffer / StringBuilder

La concaténation d'objets **String** est peu performante

Alternative pour construire des chaînes : **StringBuilder**

- chaîne modifiable

- meilleures performances

Création

- `new StringBuilder()`

- `new StringBuilder(String)`

StringBuffer / StringBuilder

Construction dynamique et fluide

`append(valeur)`

valeur : type primitif ou objet

`insert(int position, valeur)`

Conversion en String

`toString()`

StringBuffer / StringBuilder

Exemple:

```
String nom = "Martin";  
int age = 25;  
  
// Instanciation d'un StringBuilder  
StringBuilder builder = new StringBuilder();  
  
// ajout au builder de toutes les informations à concaténer.  
builder.append("L'age de M.").append(nom).append(" est :").append(age);  
  
// La méthode toString() de la classe StringBuilder retourne le résultat  
// sous forme de chaîne de caractères  
System.out.println(builder.toString());
```

Atelier (TP)

Objectifs du TP: apprendre à utiliser la classe `StringBuilder`.

Description du TP:

Dans ce TP, vous allez utiliser la classe `java.util.StringBuilder` et comparer ses performances avec l'opérateur de concaténation « `+` »

Chapitre 4

Les dates en Java (7 ou inf.)

Dates

java.util.Date

La plupart des méthodes dépréciées : existent mais ne doivent pas être utilisées car elles peuvent disparaître dans une future version du JDK

```
Date d1 = new Date();  
Date d2 = new Date(120, 6, 7, 12, 25, 30);
```



L'année commence à 1900

Le mois commence à 0

Donc la date d2 dans l'exemple ci-dessus est le 07/07/2020... !!!

*Remplacée depuis java 8 par les classes **LocalDate** et **LocalDateTime***

Calendar (1/2)

java.util.Calendar

Pour créer une date, il est recommandée de créer d'abord un calendrier, puis d'en extraire la Date.

```
Calendar cal = Calendar.getInstance();  
cal.set(2020, 6, 7, 19, 30, 56);  
  
Date date = cal.getTime();
```

Attention, le problème sur l'année a été corrigé mais pas celui sur le mois... Il s'agit toujours du 07/07/2020

Calendar (2/2)

java.util.Calendar

Il existe une méthode unitaire qui permet de ne modifier qu'une partie de la date.

```
Calendar cal = Calendar.getInstance();  
cal.set(Calendar.YEAR, 2002);  
cal.set(Calendar.MONTH, 8);  
cal.set(Calendar.DAY_OF_MONTH, 19);  
cal.set(Calendar.HOUR_OF_DAY, 0);  
cal.set(Calendar.MINUTE, 0);  
cal.set(Calendar.SECOND, 0);  
  
Date date = cal.getTime();
```

Formatage des dates et conversion

java.text.SimpleDateFormat

*conversion **Date** < = > **String***

constructeur avec un pattern de conversion.

*String → Date: **parse(String)***

*Date → String: **format(Date)***

```
SimpleDateFormat formateur = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
```

```
Date date1 = new Date();
```

```
String chaine1 = formateur.format(date1);
```

```
String chaine2 = "12/07/2020 14:25:32";
```

```
Date date2 = formateur.parse(chaine2);
```

SimpleDateFormat - pattern

Les patterns utilisables dans le constructeur de SimpleDateFormat

Pattern	Signification	Exemple
dd	Jour du mois	25
MM	Numéro du mois	08
yyyy	Année	2020
yy	Année (2 derniers chiffres)	20
HH	Heures (de 0 à 23)	15
mm	Minutes (de 0 à 59)	45
ss	Secondes (de 0 à 59)	59
EEE	Nom du jour en abrégé	Mon.
EEEE	Nom du jour	Monday
MMM	Nom du mois en abrégé	Jun.
MMMM	Nom du mois	June

Locale

java.util.Locale

La locale est une classe qui permet de définir des paramètres culturels, comme par exemple la langue et le formatage des dates.

```
Locale france = new Locale("fr", "FR");
```

ou

```
Locale france = Locale.FRANCE;
```

Seuls les gros pays ont droit à une constante (G7, Chine et quelques autres)

Locale – formatage des dates

Exemple de l'utilisation de la locale pour le formatage des dates

```
SimpleDateFormat formateur = new SimpleDateFormat("EEEE dd MMMM yyyy", Locale.FRANCE);  
System.out.println(formateur.format(new Date()));
```

mardi 07 juillet 2020

```
SimpleDateFormat formateur = new SimpleDateFormat("EEEE dd MMMM yyyy", Locale.US);  
System.out.println(formateur.format(new Date()));
```

Tuesday 07 july 2020

Locale – formatage avec caractères en dur

Il est possible d'ajouter des caractères dans le pattern de formatage à condition de les mettre entre single quotes. Exemple: 'à'.

Exemple complet:

```
SimpleDateFormat formateur = new SimpleDateFormat("'Le'EEEE dd MMMM yyyy 'à' HH'h'",  
Locale.FRANCE);  
System.out.println(formateur.format(new Date(120, 6, 7, 9, 0, 0)));
```

Le mardi 07 juillet 2020 à 09h

Atelier (TP)

Objectifs du TP: manipuler les dates et apprendre à les formater.

Description du TP:

Dans ce TP, vous allez utiliser les classes `java.util.Date`, `java.util.Calendar` et `java.text.SimpleDateFormat`.

Chapitre 5

Redéfinition de equals

La méthode equals

Définie dans la classe Object

Toute classe possède cette méthode par héritage.

Signature: boolean equals(Object object)

Problème : La méthode **equals** de la classe Object compare les adresses mémoires.

Exemple:

```
Ville v1 = new Ville("Nice", 343000);  
Ville v2 = new Ville("Nice", 343000);  
  
boolean result = v1.equals(v2); // result vaut false
```

La méthode equals

Dans ce cas la méthode equals ne peut retourner true que si 2 références pointent vers le même objet en mémoire.

Exemple:

```
Ville v1 = new Ville("Nice", 343000);  
Ville v2 = v1;  
  
boolean result = v1.equals(v2); // result vaut true
```

La méthode equals: redéfinition

Exemple:

```
public class Ville {  
  
    private String nom;  
    private int nbHabitants;  
  
    @Override  
    public boolean equals(Object object) {  
        if (!(object instanceof Ville)) {  
            return false;  
        }  
        Ville other = (Ville) object;  
        return nom.equals(other.getNom());  
    }  
}
```

Etape 1: tester que l'objet passé en paramètre est bien une ville: **instanceof**

Etape 2: une fois qu'on est sûr qu'object est une ville, on peut caster.

Etape 3: on compare les attributs qu'on souhaite comparer.

La méthode equals: problématique des attributs null

Exemple:

```
public class Ville {  
  
    private String nom;  
    private int nbHabitants;  
  
    @Override  
    public boolean equals(Object object) {  
        if (!(object instanceof Ville))  
            return false;  
        }  
        Ville other = (Ville) object;  
        return nom.equals(other.getNom());  
    }  
}
```

Que se passe-t'il si nom est null ?

NullPointerException

La méthode equals: redéfinition avec Objects

```
public class Ville {  
  
    private String nom;  
    private int nbHabitants;  
  
    @Override  
    public boolean equals(Object object) {  
        if (!(object instanceof Ville)) {  
            return false;  
        }  
        Ville other = (Ville) object;  
        return Objects.equals(nom, other.getNom());  
    }  
}
```

Avantage: Objects.equals prend en compte le cas des attributs null !!

La méthode equals pour les String

L'opérateur == est **fortement déconseillé** pour les String

Cas où l'opérateur fonctionne:

```
String v1 = "Nice";  
String v2 = "Nice";  
  
boolean result = v1 == v2; // result vaut true
```

Toutes les String créées avec l'opérateur " sont stockées dans un cache appelé String pool. Si la String existe déjà, une référence est utilisée.

Cas où il ne fonctionne pas:

```
String v1 = "Nice";  
String v2 = new String("Nice");  
  
boolean result = v1 == v2; // result vaut false
```

Atelier (TP)

Objectifs du TP: apprendre à redéfinir la méthode equals.

Description du TP:

Dans ce TP, vous allez apprendre à utiliser la méthode equals et tester son fonctionnement.

Chapitre 6

La javadoc

A quoi sert la javadoc ?

- Elle sert à documenter vos classes, vos attributs et vos méthodes
- Elle permet de fournir des informations aux personnes qui veulent comprendre ce que font vos classes.
- Elle est obligatoire en entreprise.
- L'absence de javadoc est considérée comme une **erreur de qualité de code**.
- Des outils comme SonarQube remonte une erreur pour chaque classe, attribut ou méthode non documentée

A quoi ressemble la javadoc ?

- Elle comment par `/**` et se termine par `*/`
- La commande **javadoc** permet de générer un site web static contenant la documentation de votre projet

```
/**  
 *  
 *  
 */
```

Javadoc sur les classes

- La javadoc sur la classe indique ce que fait la classe, ce qu'elle représente.
- Il est possible d'ajouter des annotations dans la javadoc comme @author ou @version
- Exemple :

```
/** Représente le concept de Ville dans l'application de recensement.  
 * Une Ville possède 2 propriétés principales :  
 * - un nom  
 * - un nombre d'habitants  
 *  
 * @author RichardBONNAMY  
 * @version 1.0  
 */  
public class Ville implements Comparable<Ville> {
```

Javadoc sur les attributs

- La javadoc sur les attributs fournit des explications sur le rôle des attributs, sur leur sens d'un point de vue fonctionnel et/ou technique.
- Exemple :

```
public class Ville implements Comparable<Ville> {  
  
    /** Nom */  
    private String nom;  
  
    /** Nombre d'habitants */  
    private int nbHabitants;  
}
```

Javadoc sur les méthodes

- La javadoc sur les méthodes fournit des explications sur le rôle d'une méthode et ce à quoi elle sert.
- Les paramètres doivent être documentés
- Exemple :

```
/** Permet de modifier la consommation de la facture en modifiant le nombre de kwh  
 * consommé pour la facture.  
 * @param nbKwh nouveau nombre de kwh consommés  
 */  
public void modifierConso(int nbKwh) {  
    this.nbKwh = nbKwh;  
    prix = this.nbKwh*0.015;  
}
```