La Programmation Orientée Objet (POO) avec Java

Module 2 – L'utilisation de classes de Java

Objectifs

- Découvrir les notions de la Programmation Orientée Objet au travers de l'utilisation d'une classe existante
- Faire la différence entre une classe et une instance

L'utilisation de classes de Java

Idée générale de la Programmation Orientée Objet

- Regrouper des variables qui « vont bien ensemble »
  - Exemple :

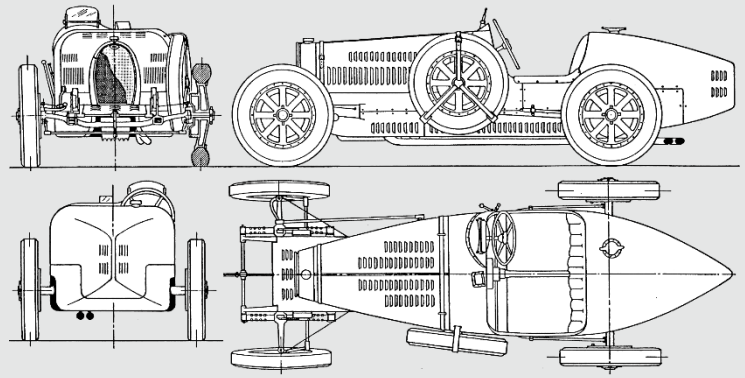
    les variables heures, minutes et secondes dans une classe Temps

- Associer les fonctions et les procédures qui manipulent ces valeurs
  - Exemple :

    les sous-algorithmes convertirEnSecondes() et changerFuseauHoraire() dans la classe Temps

Définitions

- Classe :
  - Collection d'objets caractérisés par une sémantique commune



- Instance :
  - Un élément de cette collection

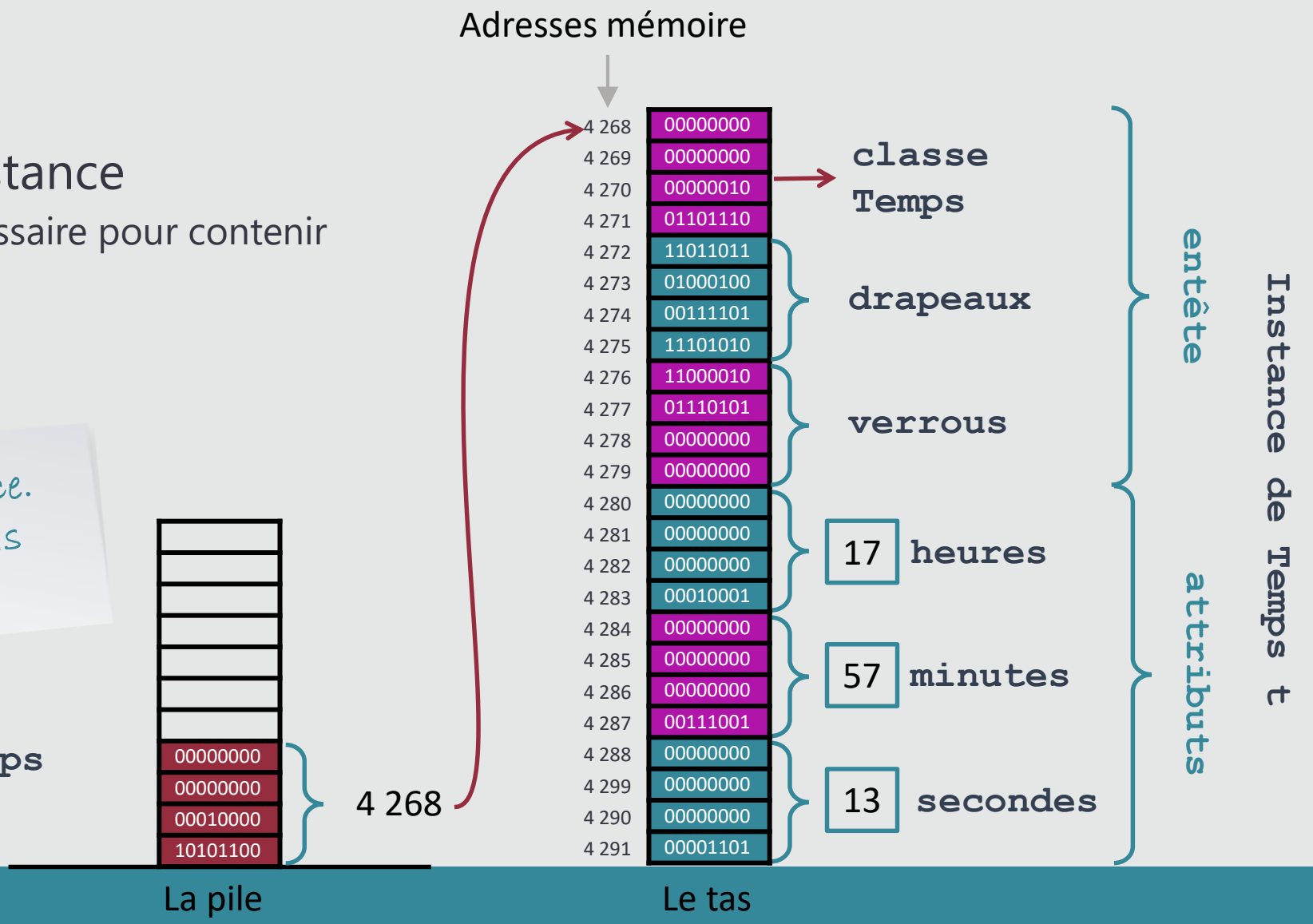
L'utilisation de classes de Java

Pour faire simple

Le constructeur

- Crée une nouvelle instance
  - Alloue la mémoire nécessaire pour contenir une nouvelle instance
  - Initialise cette instance

Une instance est un type référence.
Elle est donc instanciée sur le tas
(comme pour un tableau)



L'utilisation de classes de Java

Création d'une variable de type GregorianCalendar

```
package testGregorian;

import java.util.GregorianCalendar;

public class Test {

  public static void main(String[] args) {
    // déclaration d'une variable de type GregorianCalendar
    GregorianCalendar revo;
  }

}
```

import de la classe avec son chemin complet

Utilisation d'un constructeur de la classe GregorianCalendar

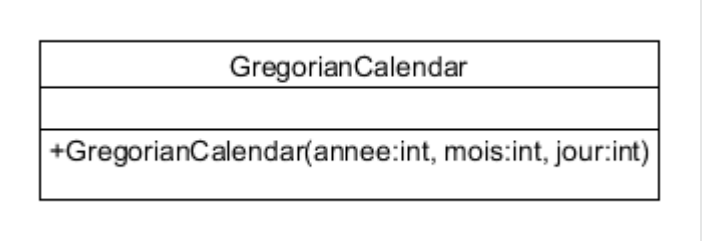
```
package testGregorian;

import java.util.GregorianCalendar;

public class Test {

  public static void main(String[] args) {
    // déclaration d'une variable de type GregorianCalendar
    GregorianCalendar revo;
    // création d'une nouvelle instance de GregorianCalendar en faisant appel au constructeur
    revo = new GregorianCalendar(1789, 7, 14);
  }

}
```



Les méthodes d'instance

- Les méthodes sont les différents services proposés par la classe
  - convertirEnSecondes() et changerFuseauHoraire() par exemple pour une classe Temps

Utilisation de méthodes de la classe GregorianCalendar

```
public static void main(String[] args) {
  GregorianCalendar revo = new GregorianCalendar(1789, 6, 14);
  System.out.format("%02d/%02d/%d%n",
  revo.get(GregorianCalendar.DAY_OF_MONTH),
  revo.get(GregorianCalendar.MONTH)+1,
  revo.get(GregorianCalendar.YEAR));
  afficherDate(revo);
  revo.add(GregorianCalendar.MONTH, 6); // 6 mois après la révolution française
  afficherDate(revo);
}

private static void afficherDate(GregorianCalendar date) {
  System.out.format("%02d %s %d%n",
  date.get(GregorianCalendar.DAY_OF_MONTH),
  date.getDisplayName(GregorianCalendar.MONTH, GregorianCalendar.LONG_FORMAT, Locale.FRANCE),
  date.get(GregorianCalendar.YEAR));
}
```

Attention n° du mois basé zéro !

GregorianCalendar
+GregorianCalendar(annee:int, mois:int, jour:int) +get(champ:int):int +getDisplayName(champ:int, style:int, local:Local):String +add(champ:int, valeur:int)

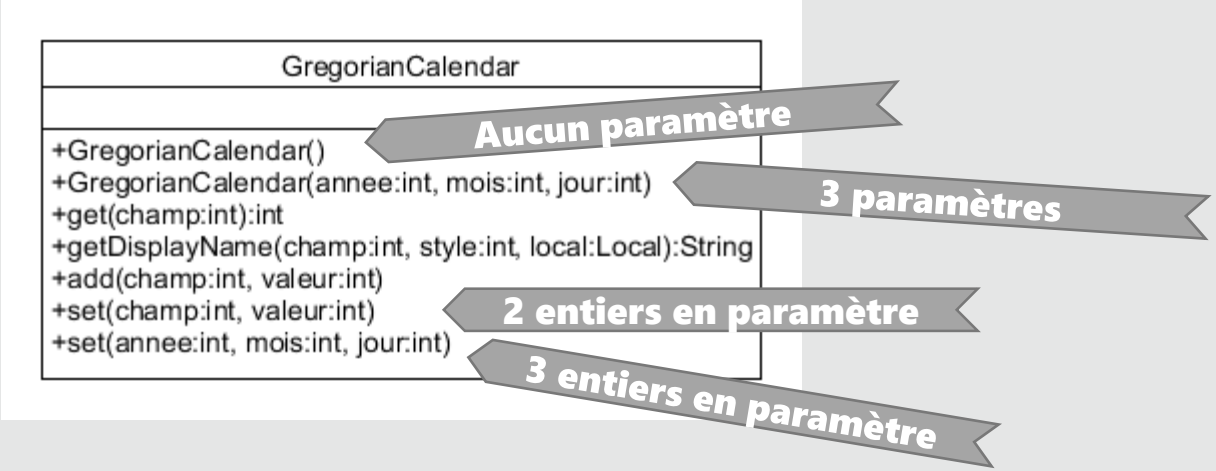
14/07/1789
14 juillet 1789
14 janvier 1790

La surcharge

- Plusieurs méthodes peuvent porter le même nom au sein d'une classe à condition qu'elles puissent être différenciées
  - Le nombre d'arguments différent
  - Les types des arguments différents

2 constructeurs

2 méthodes set



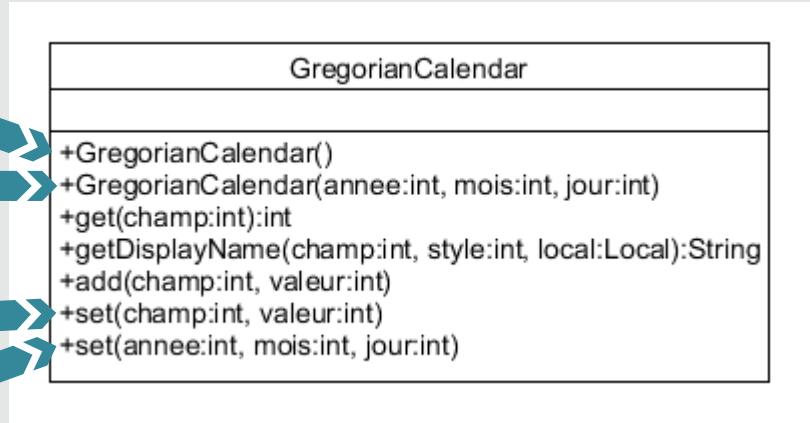
La surcharge

```
// crée une instance initialisée à la date du jour
GregorianCalendar aujourdhui = new GregorianCalendar();
afficherDate(aujourdhui);

GregorianCalendar cal =
  new GregorianCalendar(2019, GregorianCalendar.JUNE, 17);
afficherDate(cal);

cal.set(GregorianCalendar.YEAR, 2052);
afficherDate(cal);

cal.set(2017, GregorianCalendar.DECEMBER, 25);
afficherDate(cal);
```



15 novembre 2018
17 juin 2019
17 juin 2052
25 décembre 2017

Les éléments d'instance et les éléments de classe

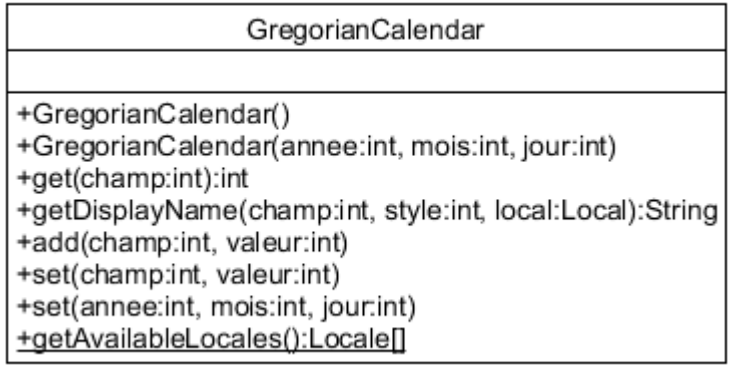
- Éléments d'instance
  - Individuel : propre à chaque instance
    - Exemples : couleur, numéro de série, prix…
- Éléments de classe
  - Collectif : concerne l'ensemble des instances
    - Exemples : prochain numéro de série, le prix le plus élevé
  - Commun : la même chose quelque soit l'instance
    - Exemples : nombre de places, nombre de volants

L'utilisation de classes de Java

Les éléments d'instance et les éléments de classe

```
Locale[] locales = GregorianCalendar.getAvailableLocales();
for(Locale l : locales)
  System.out.println(l);
```

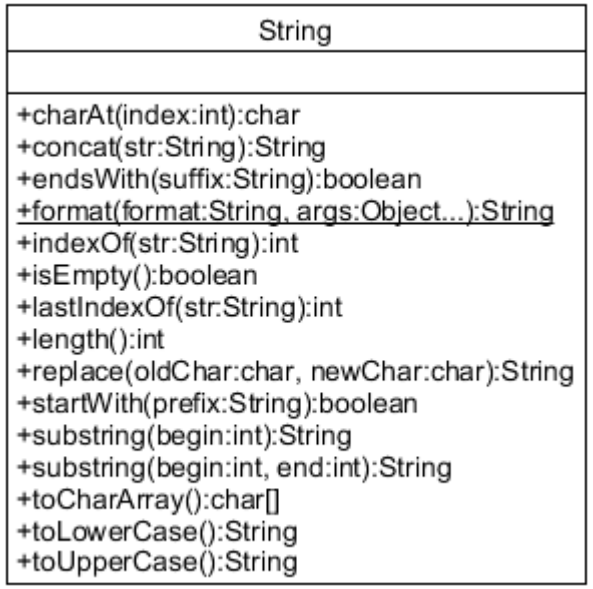
ar_AE	ko	ar_BH	cs
ar_JO	uk	pt	sr_BA_#Latn
ar_SY	lv	ar_SA	el
hr_HR	da_DK	sk	uk_UA
fr_BE	es_PR	ar_YE	hu
es_PA	vi_VN	hi_IN	fr_CH
mt_MT	en_US	ga	in
es_VE	sr_ME	en_MT	es_AR
bg	sv_SE	fi_FI	ar_EG
zh_TW	es_BO	et	…
it	en_SG	sv	



L'utilisation de classes de Java

String et StringBuilder

- La classe String est immuable
  - Aucune méthode ne modifie l'instance
  - Les méthodes telles que replace(), substring() ou toUpperCase() retourne une nouvelle instance de String



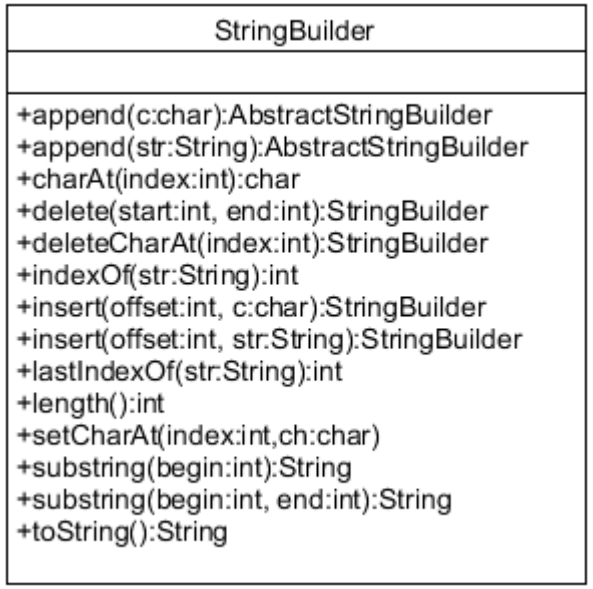

```
String nom = "Dupont";
String nomMaj = nom.toUpperCase();
String avecUnD = nomMaj.replace('T', 'D');
System.out.printf("%s %s %s%n",nom, nomMaj, avecUnD);
```

Dupont DUPONT DUPOND

L'utilisation de classes de Java

String et StringBuilder

- La classe StringBuilder est modifiable
  - Les méthodes telles que replace(), substring() ou toUpperCase() modifie l'instance.




```
StringBuilder nom = new StringBuilder("Dupont");
nom.setCharAt(5, 'd');
nom.append(" ").append("jean");
System.out.println(nom.toString());
```

Dupond jean

TP