Bibliothèque Java

Dr Khadim DRAME
kdrame@univ-zig.sn

Département Informatique
UFR Sciences et Technologies
Université Assane Seck de Ziguinchor

Juillet 2022

Objectifs du cours

- manipuler les chaînes de caractères avec la classe String
- utiliser les classes enveloppantes
- manipuler les tableaux dynamiques avec la classe ArrayList
- utiliser la classe Scanner
- utiliser les classes de manipulation de dates et de temps

Plan

Bibliothèque Java

- La bibliothèque de Java fournit de nombreuses classes
  - classe **String** pour manipuler les chaînes de caractères
  - classes **enveloppantes** pour manipuler les types primitifs
  - classe **ArrayList** très utilisée pour manipuler les tableaux dynamiques
  - classe **Scanner** pour la lecture de données
  - classes pour manipuler les dates et temps : **Date**, **Calendar**, **LocalDate**, **LocalTime**, **LocalDateTime**, **Duration**, **Period**

Plan

Classe String

- Les chaînes de caractères sont très utilisées en programmation
- En Java, la classe **String** fournit des méthodes pour les manipuler
  - création d'objets de type String
  - accès aux caractères et sous-chaines d'une chaîne
  - concaténation de chaînes de caractères
  - comparaison de chaînes de caractères
  - recherche dans une chaîne caractères

Création de chaînes de caractères

- Syntaxe

  String <nom_chaine> = **new** String("une chaine");

  String <nom_chaine> = "une chaine";

- Exemple

```
1 public class Test{
2   public static void main (String args[]){
3     String msg1 = "Bonjour la classe";
4     String msg2 = new String("Bienvenue au cours de POO");
5     System.out.println(msg1);
6     System.out.println(msg2);
7   }
8 }
```

Manipulations de chaînes de caractères

1. `int length()` retourne la longueur d'une chaîne de caractères
2. `char charAt(int i)` retourne le caractère à la $i^{ème}$ position
3. `String substring(int i, int j)` retourne la séquence de caractères entre les positions i (comprise) et j
4. `String concat(String s)` permet de concaténer une chaîne avec une autre s donnée en paramètre
5. L'opérateur `+` permet de concaténer deux chaînes de caractères

Manipulations de chaînes de caractères

1. `boolean equals(String s)` permet de tester l'égalité de 2 chaînes
2. `boolean equalsIgnoreCase(String s)` permet de tester l'égalité de 2 chaînes sans tenir compte de la case
3. `int compareTo(String s)` permet de comparer 2 chaînes
4. `boolean contains(String s)` vérifie si une chaîne contient une sous-chaîne
5. `boolean isEmpty()` vérifie si une chaîne est vide

Manipulation de chaînes de caractères

- Exemple

```java
public class Test{
  public static void main (String args[]){
    String msg1 = "Bonjour la classe";
    String msg2 = "Bonjour ";
    msg2 += "la classe";
    String msg3 = "Bienvenue !";
    System.out.println(msg1==msg2);//false
    System.out.println(msg1.equals(msg2));//true
    System.out.println(msg1.equals(msg3));//false
    String ss = msg1.substring(0, 7);
    System.out.println(ss);//Bonjour
    String msg = "Bienvenue";
    msg = msg.concat(" au cours de");
    System.out.println(msg);//Bienvenue au cours
    msg += " POO !";
    System.out.println(msg);//Bienvenue au cours de POO !
  }
}
```

Plan

Classes enveloppantes

- Java fournit des classes permettant d'envelopper la valeur d'un type primitif
- On les appelle **classes enveloppantes** ou wrapper class

Type primitif	Classe enveloppante associée
boolean	java.lang.**Boolean**
char	java.lang.**Character**
byte	java.lang.**Byte**
short	java.lang.**Short**
int	java.lang.**Integer**
long	java.lang.**Long**
float	java.lang.**Float**
double	java.lang.**Double**

Méthodes de classes enveloppantes

- `valueOf()` pour créer une instance d'une classe enveloppante
- `parseXxx(String)` pour convertir une chaîne de caractères en un type primitif (**Xxx** nom du type)
- `xxxValue()` retourne la valeur enveloppée (**xxx** nom du type)

```
1 public class Test{
2   public static void main (String args[]){
3     Integer i = Integer.valueOf(12);
4     int e = Integer.parseInt("12");//12
5     boolean b = Boolean.parseBoolean("true");//true
6     double d = Float.parseDouble("12.5");//12.5
7     int x = 1 + i.intValue();//13
8   }
9 }
```

- Les classes enveloppantes contiennent des constantes
  - **MIN_VALUE** et **MAX_VALUE** pour les entiers

Autoboxing

- Integer i = 2 ; est accepté par le compilateur comme
  Integer i = Integer.valueOf(2) ; // boxing
- Exemple

```
1 public class Test{
2   public static void main (String args[]){
3     Integer i = 10;
4     Integer j = 2;
5     Integer k = i + j;
6     int p = 2 * k;
7     System.out.println(p);//24
8   }
9 }
```

Plan

Classe ArrayList

- Les tableaux sont des structures de données très importantes
- La classe **java.util.ArrayList** permet de manipuler des tableaux dynamiques
- Pour la création d'un objet de type ArrayList, il est nécessaire de spécifier le type des éléments du tableau
- Exemple

```
1  public class Test{
2    public static void main (String args[]){
3      ArrayList<Integer> lst = new ArrayList<Integer>();
4      for(int i=1;i<=10;i++)
5        lst.add(i);
6      //Parcours d'un tableau
7      for(int i=0;i<lst.size();i++)
8        System.out.print(lst.get(i)+" ");
9    }
10 }
```

Méthodes de ArrayList

- `int size()` retourne la taille du tableau
- `boolean add(e)` ajoute l'élément **e** à la fin du tableau
- `boolean add(int i, e)` ajoute l'élément **e** au rang **i**
- `boolean remove(e)` supprime la première occurrence de l'élément **e** du tableau
- `boolean remove(int i)` supprime l'élément au rang **i** du tableau
- `get(int i)` retourne l'élément au rang **i** du tableau
- `set(int i, e)` remplace l'élément au rang **i** par **e**
- `boolean contains(e)` vérifie si le tableau contient l'élément **e**
- `boolean isEmpty(e)` vérifie si le tableau est vide

Méthodes de ArrayList

- Exemple

```java
public class Test{
  public static void main (String args[]){
    ArrayList<Integer> lst1 = new ArrayList<Integer>();
    for(int i=1;i<=10;i++)
      lst1.add(i);
    ArrayList<Integer> lst2 = new ArrayList<Integer>();
    lst2.addAll(lst1); //copie tous les éléments de lst1
    lst2.remove(0);
    lst2.remove(2);
    System.out.println(lst2.contains(1));//false
    System.out.println(lst2.contains(2));//true
    System.out.println(lst2.size());//8
    //Parcours d'un tableau avec for each
    for(int v : lst2)
      System.out.print(v+" ");//2 3 5 6 7 8 9 10

  }
}
```

Plan

Classe **Scanner**

- La classe **java.util.Scanner** permet de lire des données et de contrôler leur validité
- Elle fournit des méthodes
  - `nextXxx()` (**Xxx** nom du type) pour récupérer les valeurs saisies
  - `hasNextXxx()` pour vérifier le type de valeurs saisies
- Exemple 1

```
1  public class TestScanner{
2    public static void main (String args[]){
3      Scanner scanner = new Scanner(System.in);
4      System.out.print("Donner votre nom : ");
5      String nom = scanner.nextLine();
6      System.out.print("Donner votre âge : ");
7      int age = scanner.nextInt();
8      System.out.println("Nom : "+nom+" et âge : "+age);
9    }
10 }
```

Classe **Scanner**

- Exemple 2

```java
public class TestScanner{
  public static void main (String args[]){
    Scanner scanner = new Scanner(System.in);
    System.out.print("Donner votre nom : ");
    String nom = scanner.nextLine();
    Integer age = null;
    do {
      System.out.print("Donner vore âge : ");
      if (!scanner.hasNextInt()) {
        scanner.next();
        System.out.println("Saisir une valeur valide svp ");
        continue;
      }
      age = scanner.nextInt();
    } while(age == null);
    System.out.println("Nom : "+nom+" et âge : "+age);
  }
}
```

Plan

Classe **Date**

- **java.util.Date** est la première classe définie pour représenter une date mais
  - elle ne peut pas représenter des dates antérieures à 1900
  - elle ne supporte pas les fuseaux horaires
  - elle ne supporte que les dates du calendrier grégorien
- Une bonne partie de ses méthodes est dépréciée mais la classe **Date** est largement utilisée

Classe Date

- Exemple

```java
package sn.uasz.l3.poo;

import java.util.Date;
public class TestDate{
  public static void main (String args[]){
    Date date1 = new Date();
    //crée la date du jour avec l'heure actuelle
    Date date2 = new Date(0); //01/01/1970 00:00:00.000
    //date2.set(2022, 6, 10, 12, 0, 0);
    //on met la date du 10 juin 2022 à 12:00:00
    System.out.println(date1);
    System.out.println(date2);
    System.out.println(date1.after(date2));//true
    System.out.println(date2.before(date1));//true
    System.out.println(date1.equals(date2));//false
    System.out.println(date1.compareTo(date2));//1
  }
}
```

Classe **Calendar**

- La classe **java.util.Calendar** a été proposée pour remplacer la classe java.util.Date
  - elle permet de représenter toutes les dates
  - elle supporte les fuseaux horaires (**timezone**)
  - elle permet de gérer plusieurs calendriers
- La classe **Calendar** est plus complète que la classe **Date** mais reste moins utilisée

Classe **Calendar**

- Exemple

```
1  package sn.uasz.l3.poo;
2
3  import java.util.Date;
4  public class TestCalendar{
5    public static void main (String args[]){
6      //Date et heure actuelles, fuseau horaire du système
7      Calendar date = Calendar.getInstance();
8      //Date et heure actuelles, fuseau horaire de la France
9      Calendar dateFrance = Calendar.getInstance(Locale.FRANCE);
10     //Date et heure actuelles, fuseau horaire GMT
11     Calendar dateGmt = Calendar.getInstance(TimeZone.
       getTimeZone("GMT"));
12     System.out.println(date.toInstant());
13     date.add(Calendar.HOUR, 15);
14     //on ajoute 15 heures à la date
15     System.out.println(date.toInstant());
16     //on met la date du 12 juin 2022 à 12:00:00
17     date.set(2022, 6, 12, 12, 0, 0);
18   }
19 }
```

API **Date/Time**

- De nouvelles classes ont été introduites par Java 8 dans le package **java.time**
  - **LocalDate** pour représenter une date
  - **LocalTime** pour représenter une heure
  - **LocalDateTime** pour représenter une date et une heure
  - **Instant** pour représenter un point dans le temps
  - **Duration** pour représenter une durée
  - **Period** pour représenter une période (durée en années, mois,...)

API **Date/Time**

- Exemple 1

```
1  package sn.uasz.l3.poo;
2  import java.time.*;
3
4  public class TestDateTime{
5    public static void main (String args[]){
6      LocalDate dateActuel = LocalDate.now();
7      LocalTime heureActuel = LocalTime.now();
8      LocalDateTime dateHeureActuel = LocalDateTime.now();
9      LocalDate date = LocalDate.of(2022, Month.JUNE, 12);
10     //12/06/2022
11     LocalDateTime dateTime = date.atTime(12, 00);
12     //12/06/2022 12:00:00
13     LocalTime time = dateTime.toLocalTime();//12:00:00
14     System.out.println(date);//2022-06-12
15     System.out.println(time);//12:00
16   }
17 }
```

API **Date/Time**

- Exemple 2

```java
package sn.uasz.l3.poo;
import java.time.*;

public class TestDateTime{
  public static void main (String args[]){
    Instant debut = Instant.now();
    // traitement à mesurer
    Duration duree = Duration.between(debut, Instant.now());
    System.out.println(duree.toMillis());
    LocalDate dateDebut = LocalDate.of(2020, 05, 25);
    LocalDate dateFin = LocalDate.of(2022, 10, 15);
    Period period = Period.between(dateDebut, dateFin);
    System.out.println(period);//P2Y4M20D
    System.out.println(period.getYears());//2
    System.out.println(period.getMonths());//4
    System.out.println(period.getDays());//20
  }
}
```

API **Date/Time**

Exercice d'application

- créer une classe `Employe` caractérisée par un matricule, un nom, une date de naissance et une date d'embauche.
- définir les méthodes `getAge()` et `getAnciennete()` permettant de calculer et retourner respectivement son âge et son ancienneté.