

Plan du cours

Table des matières

1	Les chaînes de caractères	1
2	Retour vers les objets	2
2.1	Les constructeurs	2
2.2	Le mécanisme de création et d'initialisation d'un objet	6
2.3	Durée de vie des variables et des objets	7
2.4	Surcharge de méthodes	8
2.5	Variables de classe et variables d'instance	8
2.6	Accessibilité	8
2.7	Accesseurs	9
2.8	This	10

1 Les chaînes de caractères

La classe `String`

- à noter : pas obligatoirement de `new` pour créer un objet `String` ;
- une `String` est implémentée comme un tableau **constant** de caractères ;
- $\Rightarrow$ *on ne modifie pas une `String`, on crée un nouvel objet `String` (la classe `String` est une classe non modifiable).*

La classe `String`

Quelques méthodes : `int length()` `char charAt(int ind)` `boolean equals(String s)` `int indexOf(char c)` `String substring(int début, int fin)` `String substring(int début)` `String trim()`

Les classes `StringBuffer` et `StringBuilder`

Les classes `StringBuffer` et `StringBuilder` implémentent les chaînes de caractères, comme la classe `String`, à la différence qu'une instance de `StringBuffer` ou de `StringBuilder` est **modifiable**.

$\Rightarrow$ plus efficace lorsqu'on effectue beaucoup de modifications sur une chaîne de caractères.

Les classes `StringBuffer` et `StringBuilder`

- Les principales méthodes : `append`, `insert`, `setCharAt`,...
- `StringBuilder` a de meilleures performances que `StringBuffer`, mais n'est pas robuste par rapport aux accès concurrents.
- `StringBuilder` existe depuis le JDK 1.5.
- `StringBuilder` est la classe habituellement utilisée.

Exemple

On veut qu'une bibliothèque affiche tous les livres qu'elle contient :
On pourrait écrire

```
// Dans la classe Bibliotheque
public String toString(){
    String tmp = "Bibliothèque : " + nom;
    if (dernier>=0) {
        tmp += "Liste des livres :\n";
        for (int i=0;i<=dernier;i++){
            tmp += biblio[i].toString() + "\n";
        }
    }
    else
        tmp += "\n Pas de livres pour le moment!";
    return tmp;
}
```

Exemple

Mais il faut écrire :

```
// Dans la classe Bibliotheque
public String toString(){
    StringBuilder sb = new StringBuilder();
    sb.append("Bibliothèque : ");
    sb.append(nom);
    if (dernier>=0) {
        sb.append("\n Liste des livres :\n");
        for (int i=0;i<=dernier;i++){
            sb.append(biblio[i].toString());
            sb.append("\n");
        }
    }
    else
        sb.append("\n Pas de livres pour le moment!");
    return sb.toString();
}
```

2 Retour vers les objets

2.1 Les constructeurs

Les constructeurs

- par défaut, un objet est construit à l'aide de l'opérateur **new**, et d'un **constructeur par défaut** :

```

public class Personne{
    private String nom;
    private String prenom;
    private int age = 18;
}
Personne p = new Personne();

```

- Par défaut, le mécanisme d'instanciation fait comme s'il y avait un **constructeur vide** (= le constructeur par défaut) dans la classe

```

public Personne(){
}

```
- les initialisations faites dans les déclarations d'attributs sont prises en compte (ici, `p.age` vaut 18)

Les constructeurs

- On peut décider comment se construisent les objets :

```

public class Personne{
    private String nom;
    private String prenom;
    private int age = 18;
    public Personne(String nom, String prenom){
        this.nom = nom;
        this.prenom = prenom;
    }
}

```

- la présence d'un constructeur annule le mécanisme du constructeur par défaut. Maintenant :

```

Personne p = new Personne(); // NON
Personne p = new Personne("Dupont", "Jean"); // OUI

```

Les constructeurs

On peut écrire plusieurs constructeurs (qui spécifient plusieurs manières d'instancier la classe) :

```

public class Personne{
    public static final int MAJORITE = 18;
    public static final int MAX_AGE = 120;
    private String nom;
    private String prenom;
    private int age = MAJORITE;

    public Personne(String nom, String prenom){
        this.nom = nom;
        this.prenom = prenom;
    }
}

```

Les constructeurs

```

public Personne(String nom, String prenom, int age){
    this.nom = nom;
    this.prenom = prenom;
    if ((age > 0) && (age < MAX_AGE))
        this.age = age;
}
public Personne() {}
public void setNom(String nom){
    if (this.nom == null)
        this.nom = nom;
}
public void setPrenom(String prenom){
    if (this.prenom == null)
        this.prenom = prenom;
}
} //Personne

```

Pour instancier la classe Personne

```

Personne dupont = new Personne("Dupont","Jean");
// Jean Dupont a 18 ans. On ne peut plus changer
// aucune de ses caractéristiques

Personne durand = new Personne("Durand","André",52);
// André Durand a 52 ans. On ne peut plus changer
// aucune de ses caractéristiques.

Personne qqun = new Personne();
// qqun a 18 ans. On peut maintenant lui donner
// un nom et un prénom.
qqun.setNom("Dumoulin");
qqun.setPrenom("Nathalie");
// qqun s'appelle Nathalie Dumoulin et a 18 ans.
qqun.setNom("Duchemin");
// n'est pas pris en compte.

```

Les constructeurs

Les constructeurs font du contrôle sur l'initialisation des attributs. On préférera toujours éviter de dupliquer du code :

```

public class Personne {
    // déclaration d'attributs ...

    public Personne(String nom, String prenom, int age){

```

```

        this.nom = nom;
        this.prenom = prenom;
        if ((age > 0) && (age < MAX_AGE))
            this.age = age;
    }
    public Personne(String nom, String prenom){
        this(nom,prenom,MAJORITE);
    }
}

```

Les constructeurs

- En général, ce sont les constructeurs avec beaucoup d'arguments qui contiennent les instructions de contrôle sur les attributs ;
- Donc, en général, ce sont les constructeurs à peu d'arguments qui appellent les constructeurs avec beaucoup d'arguments :

```

public class Personne {
    // déclaration d'attributs ...

    public Personne(String nom, String prenom){
        // construire une personne sans donner son âge,
        // c'est faire le choix de la valeur par défaut
        // c'est comme construire une personne de 18 ans
        this(nom,prenom,MAJORITE);
    }

}

```

Les constructeurs

Mais ce n'est pas obligatoire! On aurait pu écrire :

```

public class Personne {
    // déclaration d'attributs ...

    public Personne(String nom, String prenom){
        this.nom = nom;
        this.prenom = prenom;
    }
    public Personne(String nom, String prenom, int age){
        this(nom,prenom);
        if ((age > 0) && (age < MAX_AGE))
            this.age = age;
    }
}

```

Les constructeurs

Contraintes sur l'emploi de `this()` (le constructeur) :

- il ne peut être appelé qu'à l'intérieur d'un constructeur ;
- il doit être la première instruction du constructeur dans lequel il est appelé
- il peut être appelé **en cascade** :

```
public class Personne {  
    // comme d'habitude, plus :  
    public static final char HOMME = 'M';  
    public static final char FEMME = 'F';  
  
    private char genre = FEMME;
```

Les constructeurs

```
    Personne(String nom, String prenom){  
        this.nom = nom;  
        this.prenom = prenom;  
    }  
    Personne((String nom, String prenom, int age){  
        this(nom, prenom);  
        if ((age > 0) && (age < MAX_AGE))  
            this.age = age;  
    }  
    Personne(String nom, String prenom, int age, char genre){  
        this(nom, prenom, age);  
        if ((genre == HOMME) || (genre == FEMME))  
            this.genre = genre;  
    }  
}
```

2.2 Le mécanisme de création et d'initialisation d'un objet

Le mécanisme de création et d'initialisation d'un objet

1. les variables de classe sont créées dès le premier appel à la classe (soit pour une instantiation – appel à `new`, soit par un accès direct à une variable de classe) :
 - (a) initialisées au niveau de la déclaration ;
 - (b) ou par un constructeur de statiques (plus rares) ;
2. les variables d'instance initialisées hors constructeur (à la déclaration) ;
3. exécution du constructeur ;

Exemple

```
public class FeuTricolore{
    public static final String[] COULEURS =
        {"VERT","ORANGE","ROUGE"};
    private int feu;
    public FeuTricolore(int feu){
        this.feux = feu%COULEURS.length;
    }
    public FeuTricolore(){
    }
    public String toString(){
        return COULEURS[feu];
    }
    public void tourne(){
        feu = (feu+1)%COULEURS.length;
    }
} //FeuTricolore
```

Exemple

```
public static void main(String[] args){
    System.out.println("Quelles sont les "
        + "couleurs possibles?");
    for(int i=0;i<FeuTricolore.COULEURS.length;
        i++)
        System.out.println(FeuTricolore.COULEURS[i]);
    FeuTricolore ft = new FeuTricolore(2);
    System.out.println(ft);
    for (int i=0;i<10;i++){
        ft.tourne();
        System.out.println(ft);
    }
} //main
```

2.3 Durée de vie des variables et des objets

Durée de vie des variables et des objets

- un objet existe tant qu'il existe une référence vers lui ;
- une variable d'instance d'un type primitif existe tant qu'il existe une référence vers l'objet auquel elle appartient.

```
public Personne dummy(){
    String dup = "Dupont";
    Personne dupont = new Personne(dup,"Jean",
        23,Personne.HOMME);
}
```

```

    Personne dupont2 = new Personne(dup,"Paul",
                                    23,Personne.HOMME);
    Personne durand = new Personne("Durand","Andrée",52);
    dupont = durand;
    return durand;
}

.... // ailleurs, beaucoup plus loin
Personne p = dummy();

```

2.4 Surcharge de méthodes

La surcharge de méthodes

- C'est la possibilité d'avoir *un même nom de méthode avec des signatures différentes* (et donc des comportements différents).
- C'est la nature des paramètres effectifs qui détermine la méthode exécutée.
- *Le compilateur choisit la méthode à utiliser par concordance de l'appel et de la signature.*
- Attention : **deux méthodes ne peuvent pas être différenciées uniquement par leur type de retour.**

2.5 Variables de classe et variables d'instance

Variables de classe et variables d'instance

- une **variable de classe** (**static**) est une variable qui n'est créée qu'une seule fois par classe, et qui **est partagée par toutes les instances de cette classe** ;
- une **variable d'instance** est une variable qui **existe pour chaque instance d'une classe**, et qui contient une valeur relative à l'instance à laquelle elle appartient.

Exemples :

- les constantes d'une classe
- les variables qui contiennent une information relative à l'ensemble des instances d'une classe (compteur d'instances,...)

Méthodes d'instance, méthodes de classe

Une méthode qui ne manipule aucune variable d'instance (par exemple, une méthode qui effectue un calcul uniquement à partir de ses paramètres) a vocation à être déclarée **static**.

```

public static boolean estValide(int age){
    return (age > 0 && age <= MAXAGE);
}

```

2.6 Accessibilité

Les modificateurs d'accessibilité

A l'intérieur d'une classe, une variable ou une méthode peut être définie avec un modificateur d'accès. Les différents modificateurs d'accessibilité sont :

private l'élément (variable ou méthode, d'instance ou de classe) est privé, il n'est accessible que depuis la classe elle-même (le code de la classe dans laquelle il est défini) ;

pas de modificateur d'accès l'accès est dit *package*.

protected l'accès est étendu (par rapport à **private** au code des classes du même package **et** aux sous-classes de la classe. *Nous reviendrons plus tard sur cette notion, liée à l'héritage* ;

public accessible à partir de tout code qui a accès à la classe où l'élément est défini.

2.7 Accesseurs

Accesseurs

Ajoutons un nouvel accesseur dans `Bibliotheque` :

```
private Livre[] biblio;

public Livre[] getBiblio(){ return biblio;}
```

et maintenant...

```
// ailleurs...
Bibliotheque laBu;
...
Livre[] bib = laBu.getBiblio();
for(int i=0;i<bib.length;i++)
    bib[i] = null;
```

ERREUR!!

Accesseurs

Conclusion : il faut se méfier des accesseurs en lecture et conserver cachées les propriétés qui ne concernent pas les autres...

Oui, mais alors

```
//class Personne
public String getNom(){
    return nom;
}
```

est-ce dangereux ?

Accesseurs

Non!

```
// class Test
Personne p = new Personne("Dumoulin","Isabelle",20);
String unNom = p.getNom();
unNom = unNom.toUpperCase();
// unNom contient "DUMOULIN"
// p.nom contient "Dumoulin"
```

Les instances de `String` sont non modifiables (comme les instances de classes enveloppes).

2.8 This

This

Le mot-clé *this* désigne l'instance courante.

Il peut être utilisé :

- pour accéder à un membre de l'instance `this.nom = nom` ;
- pour passer sa référence à un autre objet

This

```
public class Livre {
    // comme d'habitude, mais on ajoute un attribut :
    private Bibliotheque bib;
    public Livre(String titre, String auteur,
                  String editeur,Bibliotheque bib){
        this.bib = bib // idem pour les autres attributs
        ...
    }
}
public class Bibliotheque {
    public boolean ajouteLivre(String titre, String auteur,
                               String editeur){
        Livre bouquin = new Livre(titre,auteur,editeur,this);
        ....
    }
}
```