Algorithmique et programmation par objets

Inf F3
Licence 2 MIASHS
Université Grenoble Alpes

Jerome.David@univ-grenoble-alpes.fr

2023-2024

http://miashs-www.u-ga.fr/~davidjer/inff3/

Cours 2 – Les objets

- Notions introduites
  - **Référence** : les objets sont manipulés via des références
  - **Création** : tout objet doit être créé
  - Le cas particulier des **types primitifs**
  - La **destruction des objets** : on n'a pas besoin de détruire les objets
    - La portée des variables et objets
    - Le ramasse-miettes
  - Les **classes : attributs et méthodes**
- Les **opérateurs**

Les références

- Les objets sont manipulés via des références
  - Exemples :
    - On manipule un curseur (l'objet) via des références telles que la souris ou le clavier
    - Une télévision (l'objet) via une télécommande (la référence)

```
String s;                 // s est une référence de type String
                          // mais s n'est pas un objet

s="Licence Miashs";       // s maintenant référence un objet String
```

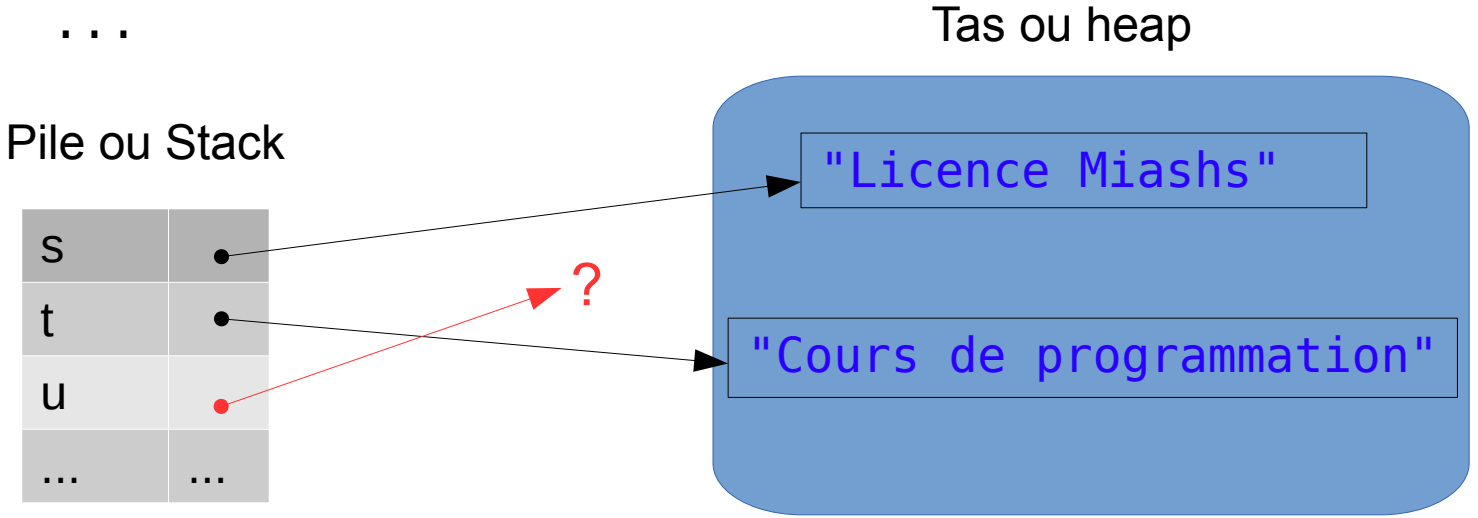
```
String s ="Licence Miashs"; // Une bonne pratique est d'initialiser
                            // systématiquement les références
```

Attention les chaînes de caractères sont un type particulier d'objets, en principe on utilise une façon plus générale pour créer des objets

La création d'objets

- Généralement, lorsque l'on déclare une référence, on l'assigne à un nouvel objet
  - Pour cela on utilise l'opérateur spécial « **`new`** »

```
String s=new String("Licence Miashs");
String t=new String("Cours de programmation");
String u=s ;
...
```

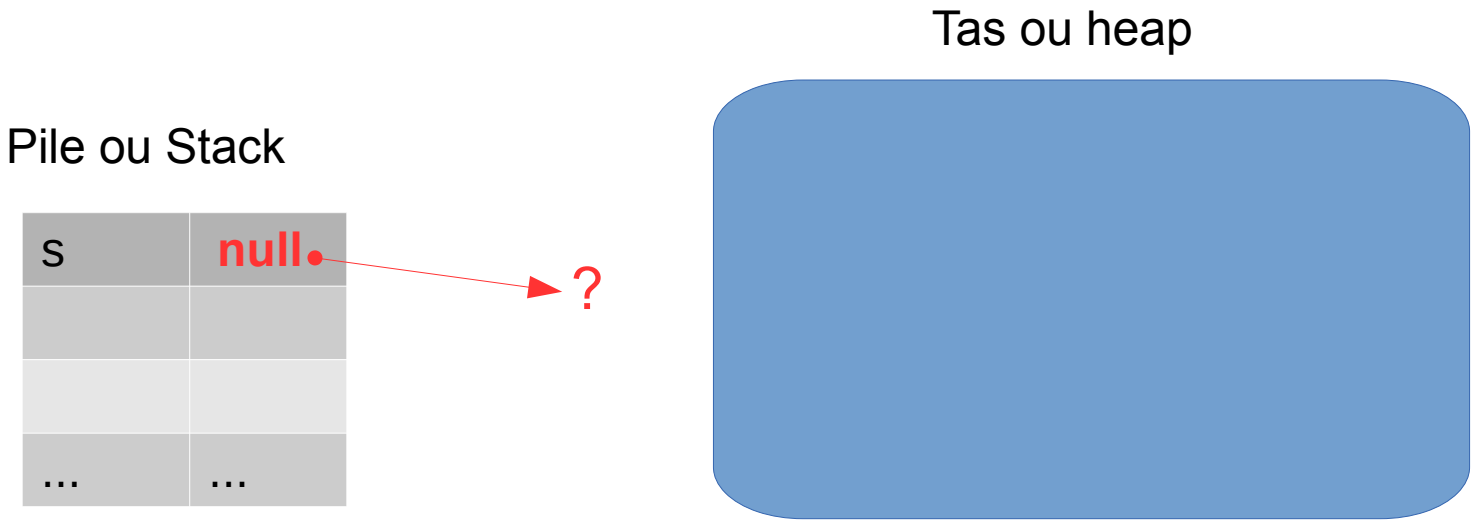


A votre avis que référence u ?

Cas des références non initialisées

- Que se passe-t-il si l'on ne créé pas l'objet ?

```
String s;
...
```

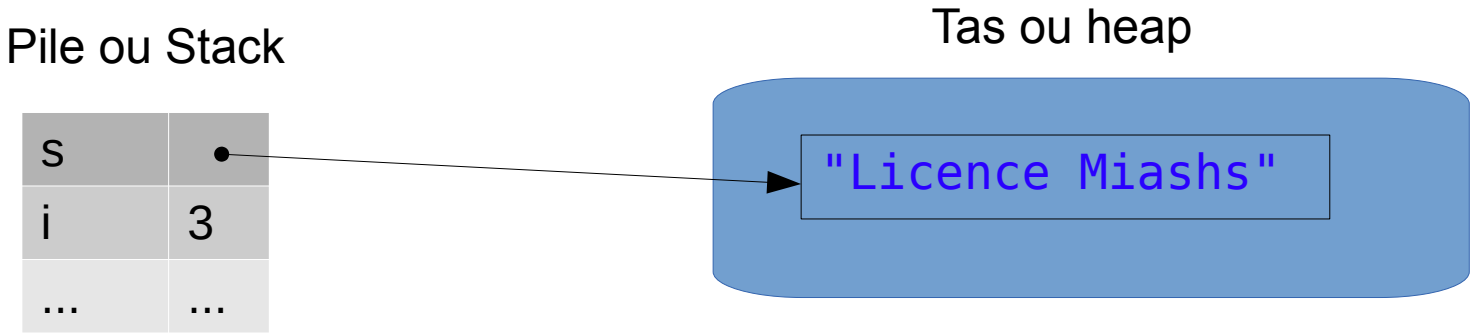


A votre avis que référence s ?

Les types primitifs

- L'allocation dynamique sur le tas est pratique mais n'est pas très efficace...
  - Java traite les types de données très fréquents, i.e. les types primitifs de manière différente
    - Pas de « **new** » mais allocation sur la pile

```
String s=new String("Licence Miashs");
int i = 3;   // i n'est pas une référence vers 3,
             // elle contient la valeur 3
...
```



Les types primitifs

- Les 8 types primitifs (ou simples)

Type	Taille (en bits)	Valeur mini.	Valeur max.	Valeur par défaut
boolean	1 ?			false
byte	8	-128	+127	0
char	16	\u0000	\uffff	\u0000
short	16	-2^15	+2^15-1	0
int	32	-2^31	+2^31-1	0
float	32	Float.MIN_VALUE	Float.MAX_VALUE	0.0f
long	64	-2^63	+2^63-1	0L
double	64	Double.MIN_VALUE	Double.MAX_VALUE	0.0d

Entiers : littéraux

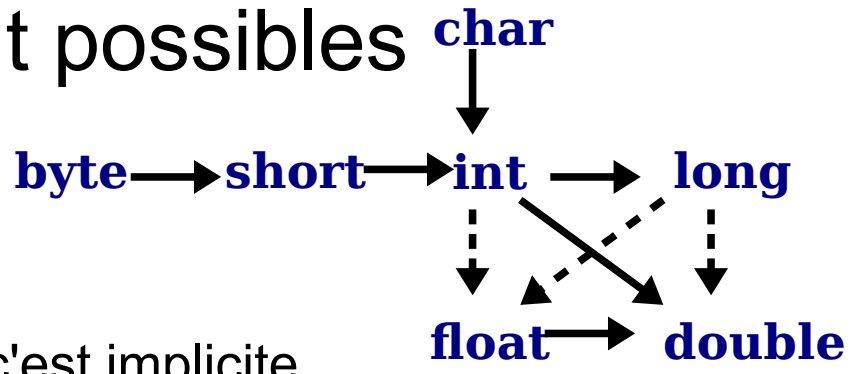
- Les entiers peuvent êtres exprimés en 3 bases
  - Décimal : `11`, `-12`, `123`, etc.
  - Octal (préfixé par 0) : `013`, `-021`, `-0173`
  - Hexadécimal (préfixé par 0x) : `0xB`, `-0x11`, `0x7B`
- Par défaut le type d'une valeur littérale entière est `int`.
  - Pour obtenir un `long`, on suffixe par `L` ou `l`
    - `11L`, `-021l`, `0x7BL`, `0l`

Réels : littéraux

- Notation décimale ou exponentielle
  - Décimale : `12.34, .123`
  - Exponentielle : `1.23E-34, .5E2`
- Par défaut, le type d'une valeur littérale réelle est double.
  - Expliciter le type float → suffixer par `f` ou `F`
    - `11.5f, -0.21F`
  - Expliciter le type double → suffixer par `d` ou `D`
    - `0.00004d, .07D`

Les conversions de type

- 2 sortes de conversions sont possibles



  - Élargissante (promotion) :
    - On va d'un type plus « large ».
      - Pas de perte d'information donc c'est implicite
      - float vers double, long vers double, int vers long, float ou double, short vers int, long, float et double
  - Restrictive (transtypage, cast)
    - On va vers un type plus restrictif, donc c'est explicite
    - La valeur est tronquée (pas d'arrondi)

```
double d = 1.7;
 int i = (int) d;
```

Les wrappers de types primitifs

- Java permet tout de même de représenter des types primitifs par des objets
  - Exemple :

```
int e = 33;
Integer f = new Integer(e);
Integer g = new Integer(33);
```

- L'autoboxing (à partir de Java 5) permet de faire la conversion automatiquement
  - Exemple :

```
Integer g = 33;
int e = g;
```

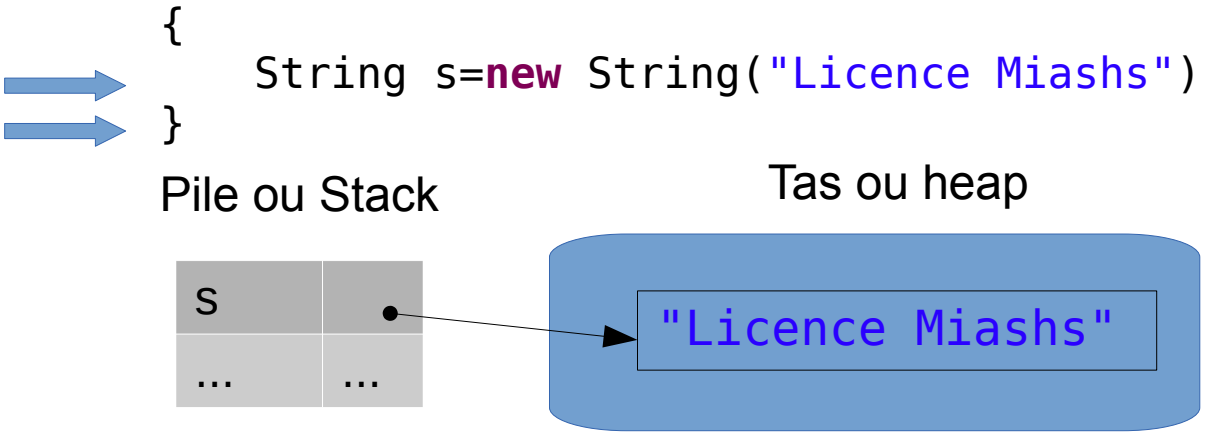
La portée des variables

- La « visibilité » d'une variable et sa durée de vie est liée au concept de portée
- La portée est déterminée en Java par des accolades { et } (qui définissent un bloc)
  - À la sortie d'un bloc toutes les variables déclarées depuis l'entrée dans ce bloc sont dépilées

```
{
    int i=3;
    // i est visible
    {
        int j=4;
        //i et j sont visibles
    }
    // seulement i est visible
}
// ni i, ni j sont visibles
```

La portée des objets

- Les objets n'ont pas la même durée de vie que les types primitifs
  - Lors de la sortie d'un bloc, seules les références déclarées dans ce bloc sont supprimées.
  - L'objet est toujours en mémoire



A la sortie du bloc, l'objet « Licence Miashs » reste en mémoire même si il n'est plus référencé

La destruction d'objets

- La mémoire n'est pas infinie, il faut donc détruire les objets lorsqu'on n'en a plus besoin
- Gérer la destruction manuellement est source de bugs
  - Fuite de mémoire, accès à des objets détruits
- Heureusement, Java s'occupe de cela à la place du programmeur !!!
  - C'est le ramasse-miettes, un programme « caché » fonctionnant en parallèle qui fait cela.
  - Le ramasse-miettes supprime les objets créés par new qui ne sont plus référencés (ou visibles, hors de portée)

Les classes

- Les classes servent à décrire des types d'objets. Elles définissent :
  - Les données q'un objet du type contient : **attributs**
  - Les comportements qu'un objet du type peut avoir : **méthodes**
- En java la définition d'une classe est de la forme suivante :

```
class MaClasseVide {
    /*
     corps de la classe
     les attributs et méthodes sont déclarés dans ce bloc
    */
}
```

- Et elle est instanciée comme cela :

```
MaClasseVide v = new MaClasseVide();
/* On dit que : v est une instance de MaClasseVide ou v est un objet
du type MaClasseVide */
```

Les attributs

- Une classe définie les attributs des objets
  - Les attributs sont des types primitifs ou des références vers d'autres objets
- Chaque instance stocke les valeurs de ses attributs
  - Il ne sont pas partagés entre les instances de la classe

```
class Ampoule {
    boolean allumee;
    int intensite;
}
```

Les attributs

- Les attributs sont accessibles via :
  - NomRefInstance.nomAttribut

```
Ampoule a = new Ampoule();

/* Les attributs d'une instance sont accessibles et peuvent être
 * modifiés
 */
a.allumee=true;
a.intensite=50;

Ampoule b = new Ampoule() ;
/* A votre avis quelles sont les valeurs par défaut ? */
System.out.println("Etat de l'ampoule : "+b.allumee);
System.out.println("Intensité de l'ampoule : "+b.intensite);
```

Les méthodes

- Les méthodes déterminent les messages qu'un objet peut recevoir
- Elles sont appelées fonction dans d'autre langages
- Les constituants principaux d'une méthode :
  - Son nom
  - La liste de ses arguments
    - types et noms des arguments passés en paramètre lors d'un appel
  - Son type de retour
    - type de la valeur en retour après un appel à la méthode
  - Le corps de la méthode
- Le nom + la liste des arguments est la signature de la méthode
  - La signature est l'identifiant de la méthode

```
TypeDeRetour nomDeMethode( /* arguments */ ) {
  /* corps de la méthode */
}
```

Les méthodes

- Une méthode est un constituant d'une classe
  - On ne peut pas créer de méthode en dehors d'une classe
- Une méthode est appelée que sur un objet
  - Il existe un cas particulier des méthodes de classes étudié plus tard
- L'appel d'une méthode sur un objet :

```
unObjet.nomDeMethode(arg1,arg2, etc.);
```

```
TypeDeRetour x = unObjet.nomDeMethode(arg1,arg2, etc.);
```

C'est ce que l'on appelle l'envoi de message en programmation par objet

La liste d'arguments

- La liste des arguments d'une méthode spécifie
  - Le type des références d'objets passés
  - Le nom que l'on donne à ces références
- Exemple d'une méthode avec paramètres et retour

```
class Point {
    double x;
    double y;

    double distance(Point p) {
        double dcarre = (x-p.x)*(x-p.x)+(y-p.y)*(y-p.y);
        return Math.sqrt(dcarre); // fonction de calcul de la racine carré
    }
}
```

Retour de méthode

- Le retour de méthode est réalisé grâce à l'instruction `return`
- `return` permet de faire deux choses :
  - Renvoyer le résultat suivant l'instruction `return`
  - Quitter la méthode
- Méthode sans retour : `void`
  - Dans ce cas, l'instruction `return` est optionnel et sert juste à quitter la méthode

```
class Point {
    double x;
    double y;

    void translate(double dx, double dy) {
        x=x+dx;
        y=y+dy;
    }
}
```

Les attributs et méthodes de classe

- Par défaut, les données (attributs) et comportements (méthodes) sont attachés aux objets et non aux classes
- Il arrive parfois que l'on ait besoin de stocker des données ou réaliser des traitements au niveau de la classe
- Java permet de déclarer
  - des attributs de classe (ou attributs statiques)
  - des méthodes de classe (ou méthodes statiques)

Cela est fait via le mot clé `static` *placé devant la déclaration d'un attribut ou d'une méthode*

Attributs de classe

- Un attribut de classe n'est pas lié à une instance particulière de la classe
  - Accessible même si aucun objet de cette classe n'a été instanciée

```
class UneClasse {
    static int i=0;
}
```

A votre avis quelles sont les valeurs de `c.i`, `d.i` et `UneClasse.i` après l'exécution de ce code ?

```
UneClasse c = new UneClasse();
UneClasse d = new UneClasse();

c.i=34;
d.i=85;
UneClasse.i=21;
```

Elles sont toutes égal à 21.

Il faut noter que même si la notation `c.i` et `d.i` sont possibles, il est préférable d'utiliser `UneClasse.i`

Méthodes de classe

- Les méthodes de classe suivent la même logique que les attributs de classe
  - On ajoute `static` devant la définition de la méthode

```
class UneClasse {
    static int i=0;

    static void incremente() {
        i++;//idem que i=i+1;
    }
}
```

```
UneClasse c = new UneClasse();
UneClasse d = new UneClasse();

c.incremente();
d.incremente();
UneClasse.incremente();
```

Même question que le slide précédent

A votre avis quelles sont les valeurs de `c.i`, `d.i` et `UneClasse.i` après l'exécution de ce code ?

Elles sont toutes égal à 3.

De même que pour les attributs d'instance, il est préférable d'utiliser `UneClasse.incremente()`

Liste des notions abordées

- Objet / type primitifs
  - Référence / valeur
  - Allocation sur le tas / allocation sur la pile
  - Visibilité des variables, ramasse-miette
- Classes
  - Attributs
  - Méthodes
    - Liste d'argument, type de retour
  - Attributs et méthodes de classes (statiques)

Compléments : Conventions de codage

- Il existe des conventions de codage en java qu'il faut s'efforcer de suivre **http://www.oracle.com/technetwork/java/codeconvtoc-136057.html**
  - Classes
    - Commencent par une majuscule
    - La première lettre de chaque mot est en majuscule
    - Exemple : `MaClasseBienNommee`
  - Pour le reste (attributs, variables, méthodes)
    - Commencent par une minuscule
    - La première lettre de chaque mot est en majuscule
    - Exemple : `unAttributBienNommeEgalement`
- Dans tous les cas
    - Pas d'accents, pas de caractères spéciaux