Master 2 LITL
Programmation pour le TAL (SLT0905V)
Éléments de syntaxe Java

Franck Sajous/CLLE-ERSS

Séance 2

CLLE ERSS UMR 5263
COGNITION, LANGUES, LANGAGE, ERGONOMIE

http://fsajous.free.fr/

Éléments de syntaxe et conventions

- les instructions se terminent par le caractère ; (obligatoire)
- le nom des variables commence par le caractère _ ou par une lettre (obligatoire)
- un nom de variable est composé des caractères lettre, chiffre et _ (obligatoire) (pas d'espace, pas de tiret -, etc. Évitez les accents)
- par convention, un nom de variable « standard » est en minuscules. On utilise une majuscule comme séparateur de mots : `monJoliMessage` (camelCase)
- tout est sensible à la casse (noms de variables, de membres, de méthodes, etc.)
- par convention, un nom de classe commence par une majuscule, le reste en minuscules (sauf séparateurs évenruels de mots → camelCase)
  Nom de classe = nom au singulier (on verra pourquoi d'un point de vue de la modélisation)
- mots-clé du langage (*while, for, if, else, return...*, *int, double*, etc.) interdits comme noms de variables, de membres, de méthodes.
- structure de blocs : les instructions d'une méthode, celles d'une boucle ou d'une condition sont délimitées par des accolades ouvrantes et fermantes { et }.
  Idem pour la définition d'une classe.

Variables et objets : déclaration et création

- en Java, on déclare toute variable avant son utilisation
- On ne peut pas écrire d'emblée :

```
message = "je suis un objet";
```

sans avoir *déclaré* la variable `message` :

```
String message;
```

1. « *je déclare que* `message` *est une variable* »
2. « *je déclare que cette variable est de type* `String` »
   (String = chaîne de caractère)

- affectation :

```
message = "je suis un objet";
```

on *affecte* la valeur "je suis un objet" à la variable `message`

- affectation + initialisation :

```
String message = "je suis un objet";
```

Java, un langage fortement typé

Problème de l'absence de typage

Exemple 1 (langage quelconque non fortement typé) :

```
maVariable1 = "Je suis une chaine";
maVariable2 = 3;
afficher (maVariable1 + maVariable2);
```

Quel affichage?

Java, un langage fortement typé

Problème de l'absence de typage

Exemple 1 (langage quelconque non fortement typé) :

```
maVariable1 = "Je suis une chaine";
maVariable2 = 3;
afficher (maVariable1 + maVariable2);
```

Quel affichage?

- "Je suis une chaîne3" ? *(conversion entier 3 → chaîne de caractères "3") et interprétation de + comme opérateur de concaténation ?*

Java, un langage fortement typé

Problème de l'absence de typage

Exemple 1 (langage quelconque non fortement typé) :

```
maVariable1 = "Je suis une chaine";
maVariable2 = 3;
afficher (maVariable1 + maVariable2);
```

Quel affichage?

- "Je suis une chaîne3" ? *(conversion entier 3 → chaîne de caractères "3") et interprétation de + comme opérateur de concaténation ?*
- 21 ? (= 18 + 3, avec 18 = lg de la chaîne et + = opérateur arithmétique ?)

Java, un langage fortement typé

Problème de l'absence de typage

Exemple 1 (langage quelconque non fortement typé) :

```
maVariable1 = "Je suis une chaine";
maVariable2 = 3;
afficher (maVariable1 + maVariable2);
```

Quel affichage?

- "Je suis une chaîne3" ? *(conversion entier 3 → chaîne de caractères "3") et interprétation de + comme opérateur de concaténation ?*
- 21 ? (= 18 + 3, avec 18 = lg de la chaîne et + = opérateur arithmétique ?)
- "Je suis une chaîne50" ? (50 = code ascii du caractère '3')

Java, un langage fortement typé

Problème de l'absence de typage

Exemple 1 (langage quelconque non fortement typé) :

```
maVariable1 = "Je suis une chaine";
maVariable2 = 3;
afficher (maVariable1 + maVariable2);
```

Quel affichage?

- "Je suis une chaîne3" ? *(conversion entier 3 → chaîne de caractères "3") et interprétation de + comme opérateur de concaténation ?*
- 21 ? (= 18 + 3, avec 18 = lg de la chaîne et + = opérateur arithmétique ?)
- "Je suis une chaîne50" ? (50 = code ascii du caractère '3')

Exemple 2 (langage quelconque non fortement typé) :

```
maVariable1 = 3.7;
maVariable2 = 2;
afficher (maVariable1 + maVariable2);
```

Quel affichage ?

Java, un langage fortement typé

Problème de l'absence de typage

Exemple 1 (langage quelconque non fortement typé) :

```
maVariable1 = "Je suis une chaine";
maVariable2 = 3;
afficher (maVariable1 + maVariable2);
```

Quel affichage?

- "Je suis une chaîne3" ? *(conversion entier 3 → chaîne de caractères "3") et interprétation de + comme opérateur de concaténation ?*
- 21 ? (= 18 + 3, avec 18 = lg de la chaîne et + = opérateur arithmétique ?)
- "Je suis une chaîne50" ? (50 = code ascii du caractère '3')

Exemple 2 (langage quelconque non fortement typé) :

```
maVariable1 = 3.7;
maVariable2 = 2;
afficher (maVariable1 + maVariable2);
```

Quel affichage ?

- 5.7 ? *Convers. entier 2 → flottant 2.0 et opér. '+' sur les flottants : 3.7 + 2.0*

Java, un langage fortement typé

Problème de l'absence de typage

Exemple 1 (langage quelconque non fortement typé) :

```
maVariable1 = "Je suis une chaine";
maVariable2 = 3;
afficher (maVariable1 + maVariable2);
```

Quel affichage?

- "Je suis une chaîne3" ? *(conversion entier 3 → chaîne de caractères "3") et interprétation de + comme opérateur de concaténation ?*
- 21 ? (= 18 + 3, avec 18 = lg de la chaîne et + = opérateur arithmétique ?)
- "Je suis une chaîne50" ? (50 = code ascii du caractère '3')

Exemple 2 (langage quelconque non fortement typé) :

```
maVariable1 = 3.7;
maVariable2 = 2;
afficher (maVariable1 + maVariable2);
```

Quel affichage ?

- 5.7 ? *Convers. entier 2 → flottant 2.0 et opér. '+' sur les flottants : 3.7 + 2.0*
- 5 ? *Conversion « dégradante » flottant 3.7 → entier 3 (suppression de la partie décimale) et opérateur '+' sur les entiers*

Java, un langage fortement typé

Problème de l'absence de typage

Exemple 1 (langage quelconque non fortement typé) :

```
maVariable1 = "Je suis une chaine";
maVariable2 = 3;
afficher (maVariable1 + maVariable2);
```

Quel affichage?

- "Je suis une chaîne3" ? *(conversion entier 3 → chaîne de caractères "3") et interprétation de + comme opérateur de concaténation ?*
- 21 ? (= 18 + 3, avec 18 = lg de la chaîne et + = opérateur arithmétique ?)
- "Je suis une chaîne50" ? (50 = code ascii du caractère '3')

Exemple 2 (langage quelconque non fortement typé) :

```
maVariable1 = 3.7;
maVariable2 = 2;
afficher (maVariable1 + maVariable2);
```

Quel affichage ?

- 5.7 ? *Convers. entier 2 → flottant 2.0 et opér. '+' sur les flottants : 3.7 + 2.0*
- 5 ? *Conversion « dégradante » flottant 3.7 → entier 3 (suppression de la partie décimale) et opérateur '+' sur les entiers*
- autre chose ?

Java, un langage fortement typé

En Java

```
double maVariable1 = 3.7;
double maVariable2 = 2;
double maVariable3 = maVariable1 + maVariable2
```

OK : ma variable2 déclarée comme double, 2 = 2.0

Java, un langage fortement typé

En Java

```
double maVariable1 = 3.7;
double maVariable2 = 2;
double maVariable3 = maVariable1 + maVariable2
```

OK : ma `variable2` déclarée comme `double`, $2 = 2.0$

```
double maVariable1 = 3.7;
int maVariable2 = 2;
double maVariable3 = maVariable1 + maVariable2
```

OK : conversion implicite non dégradante `int` $\rightarrow$ `double` de `maVariable2`

Java, un langage fortement typé

En Java

```
double maVariable1 = 3.7;
double maVariable2 = 2;
double maVariable3 = maVariable1 + maVariable2
```

OK : ma variable2 déclarée comme double, 2 = 2.0

```
double maVariable1 = 3.7;
int maVariable2 = 2;
double maVariable3 = maVariable1 + maVariable2
```

OK : conversion implicite non dégradante int → double de maVariable2

```
double maVariable1 = 3.7;
int maVariable2 = 2;
int maVariable3 = maVariable1 + maVariable2
```

Type mismatch: cannot convert from double to int

```
double maVariable1 = 3.7;
int maVariable2 = 2;
int maVariable3 = (int) maVariable1 + maVariable2
```

OK: Transtypage (cast) = conversion **explicite** double → int (3.7 → 3)
Manière du développeur de dire au compilateur *« je sais ce que je fais, je suis conscient que j'effectue une conversion dégradante. »*

Types primitifs et opérateurs

Les nombres

- `int` : nombres entiers compris entre ≈ -2 milliards et 2 milliards
- `long` : idem, mais permet de coder des nombres plus grands
- `float` et `double` : nombre réels, à virgule flottante
- priorité des opérateurs et parenthésage :
  - 2 + 3 * 5 vaut 17
  - (2 + 3) * 5 vaut 25

```
int monNombre = 3;
int autreNombre = 2;
monNombre = monNombre + 2;
monNombre = monNombre * autreNombre;

double abcd = 523.46;
double def  = abcd / 3.2;
```

Types primitifs et opérateurs (2)

Les caractères : char

```
char monCaractere = 'Z';
```

Les chaînes de caractères : String

- concaténation entre deux chaînes par l'opérateur +
- conversion implicite d'un caractère en chaîne de caractère ('c' → "c")

```
String maChaine = "la jolie ";
String chaine2 = "chain";
char caract = 'e';
String chaineConcat = maChaine + chaine2 + caract;
System.out.println (chaineConcat);
```

affiche : *la jolie chaine*

Note : tout peut être converti implicitement (sans cast) en chaîne de caractères.
On verra pourquoi ultérieurement.

Types primitifs et opérateurs (3)

Les booléens

- `boolean` : type à deux valeurs, `true` et `false`
- opérateurs logiques :
  - && : ET logique
  - || : OU logique
  - ! : négation logique

```
String maChaine = "la jolie chaine";
boolean chaineVide = maChaine.isEmpty ();
System.out.println ("chaine vide : " + chaineVide);
System.out.println ("negation : " + !chaineVide);
boolean commenceParLa = maChaine.startsWith ("la");
System.out.println ("commence par la : " + commenceParLa);
System.out.println ("vide ET commence par la : "
                    + (chaineVide && commenceParLa));
```

Opérateurs de comparaison

Les opérateurs == (égal) et != (différent de) peuvent être appliqués sur des nombres et des booléens (appliqués à des objets, ils ont une autre sémantique).
Les opérateurs > (supérieur à) et < (inférieur à) peuvent être appliqués à des nombres.
Si on déclare `int nombre = 2;`, on a les expressions `(nombre > 3)`, `(nombre == 3)` et `(nombre != 3)` qui valent respectivement `false`, `false` et `true`.

Types primitifs et classes enveloppes

En programmation objets...

les types primitifs ne devraient pas exister.

Java : un langage... « presque 100% objet », mais pas tout à fait.

Classes enveloppes (*wrappers*)

int	Integer
double	Double
char	Char
boolean	Boolean

etc.

```
int a = 6;
Integer b = new Integer(2);
b = a; // boxing

Integer c = new Integer(5);
int d = 3;
d = c; // unboxing

Double nb = new Double (5.7);
int entier = nb.intValue(); // conv. explicite
```

Choix

if

```
if (condition)
{
   instructions si vrai
}
```

Exemple :

```
if ((2 + 2) == 4)
   System.out.println ("tout va bien");
```

if/else

```
if (condition)
{
  instructions si vrai
}
else if (autre condition)
{
  instr. si autre cond. vraie
}
else
{
  instructions sinon
}
```

Exemple :

```
if ((2 + 2) == 4)
  System.out.println ("tout va bien");
else if ((2 + 2) == 5)
  System.out.println ("ça fait beaucoup");
else
  System.out.println ("revoir arithmetique");
```

Boucle *Tant que*

while

```
while (condition)
{
   instructions
}
```

Exemple :

```
int nombre = 3;
while (nombre >= 0)
{
    System.out.println (nombre);
    nombre--;
}
```

Affiche :
3
2
1
0

Boucle *Tant que*

while

```
while (condition)
{
   instructions
}
```

Exemple :

```
int nombre = 3;
while (nombre >= 0)
{
    System.out.println (nombre);
    nombre--;
}
```

Affiche :
3
2
1
0

Existe aussi en version :

```
do
{
  instructions
}
while (condition);
```

Boucle *Pour*

for

```
for (initialisation; condition; instruction)
{
   instructions
}
```

Exemple :

```
int i;
for (i = 1; i < 4; i++)
{
    System.out.println (i);
}

for (i = 3; i > 0; i--)
{
    System.out.println (i);
}
```

Affiche :
1
2
3
3
2
1

Boucle *Pour*

for

```
for (initialisation; condition; instruction)
{
   instructions
}
```

Exemple :

```
int i;
for (i = 1; i < 4; i++)
{
    System.out.println (i);
}

for (i = 3; i > 0; i--)
{
    System.out.println (i);
}
```

Affiche :
1
2
3
3
2
1

Équivalent :

```
int i = 1;
while (i < 4)
{
    System.out.println (i);
    i++;
}

i = 3;
while (i > 0)
{
    System.out.println (i);
    i--;
}
```

Autres strucutres de contrôle et tabous

Il existe, comme en C notamment, une structure `switch/case`, que nous n'utiliserons pas : `if/else if/else` équivalent et plus lisibles.

Il existe des instructions que nous nous interdirons (sous peine de représailles...) : `break` et `continue`
Il est toujours possible (et plus lisible) de s'en passer.

Autres strucutres de contrôle et tabous

Il existe, comme en C notamment, une structure `switch/case`, que nous n'utiliserons pas : `if/else if/else` équivalent et plus lisibles.

Il existe des instructions que nous nous interdirons (sous peine de représailles...) : `break` et `continue`
Il est toujours possible (et plus lisible) de s'en passer.

et pourquoi pas `goto` ?

Hello world

En Perl

```
print "Hello World";
```

Hello world

En Perl

```
print "Hello World";
```

En Java

```
public class HelloWorld
{
    public static void main (String args[])
    {
        System.out.println("Hello world !");
    }
}
```

Hello world

En Perl

```
print "Hello World";
```

En Java

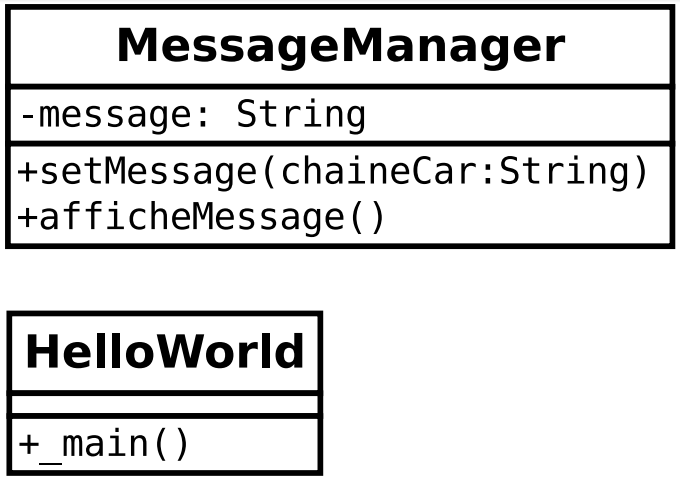
```
public class HelloWorld
{
    public static void main (String args[])
    {
        System.out.println("Hello world !");
    }
}
```

Commentaires... tout ça pour ça !

- Tout est objet : même pour afficher *bonjour*, on crée une classe
- `main()` est la méthode principale de la classe, celle qui s'exécute
- Méthode pas obligatoire dans toutes les classes, mais exécution d'un programme ⇒ au moins une méthode `main` (« point d'entrée »)
- `public` : modifieur d'accès à une classe, une méthode ou un attribut
- `static` : on verra plus tard, String args[] aussi
- `System.out.println()` : affiche une chaîne de caractères qu'on lui passe en argument (explication plus exacte à venir)

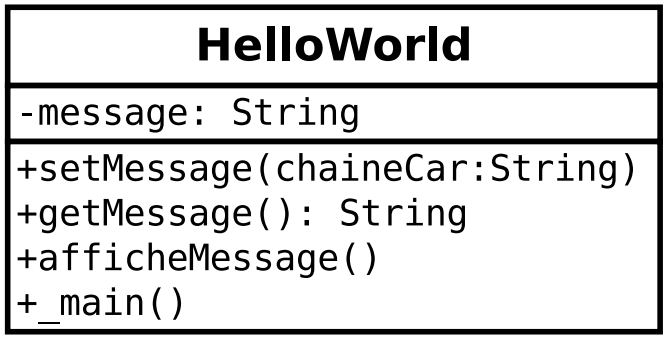
Hello world, version déléguée

```java
public class MessageManager
{
    private String message;

    public void setMessage (String chaineCar)
    {
        message = chaineCar;
    }

    public void afficheMessage ()
    {
        System.out.println(message);
    }
}


public class HelloWorld
{
   public static void main (String args[])
   {
       MessageManager messMan = new MessageManager();
       messMan.setMessage("Hello");
       messMan.afficheMessage();
   }
}
```



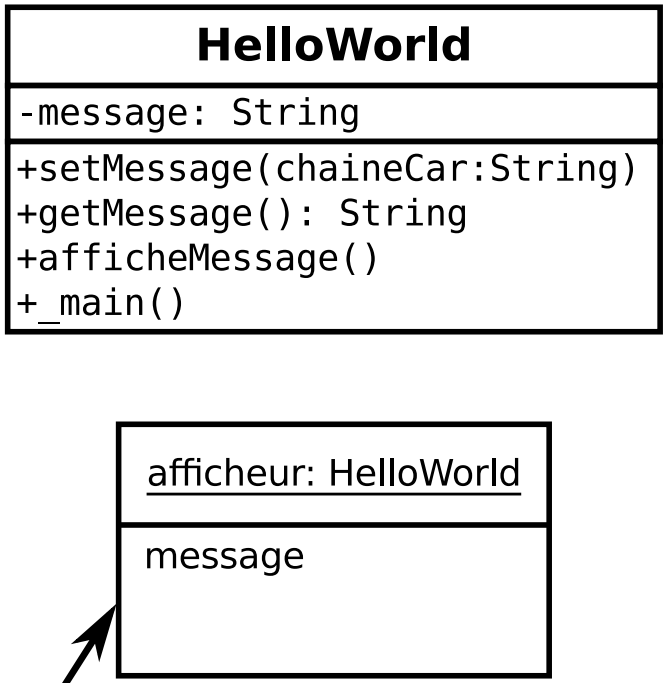
Hello world, une seule classe

```
public class HelloWorld
{
    private String message;

    public void setMessage (String chaineCar)
    {
        message = chaineCar;
    }

    public String getMessage ()
    {
        return message;
    }

    public void afficheMessage ()
    {
        System.out.println(message);
    }

    public static void main (String args[])
    {
        HelloWorld afficheur = new HelloWorld();
        afficheur.setMessage ("Je suis un objet");
        afficheur.afficheMessage();
    }
}
```



Hello world, une seule classe

```
public class HelloWorld
{
    private String message;

    public void setMessage (String chaineCar)
    {
        message = chaineCar;
    }

    public String getMessage ()
    {
        return message;
    }

    public void afficheMessage ()
    {
        System.out.println(message);
    }

    public static void main (String args[])
    {
        HelloWorld afficheur = new HelloWorld();
        afficheur.setMessage ("Je suis un objet");
        afficheur.afficheMessage();
    }
}
```



Hello world, une seule classe

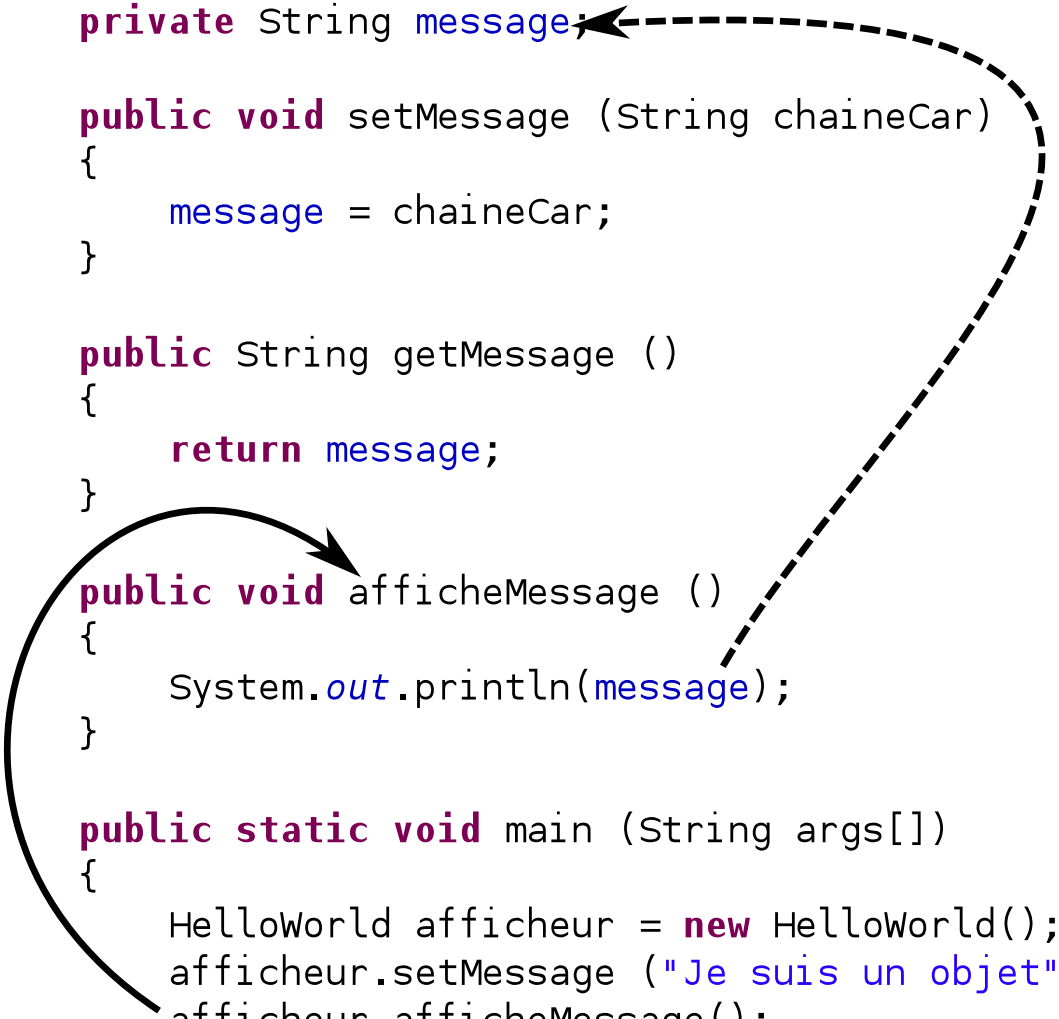
```
public class HelloWorld
{
    private String message;

    public void setMessage (String chaineCar)
    {
        message = chaineCar;
    }

    public String getMessage ()
    {
        return message;
    }

    public void afficheMessage ()
    {
        System.out.println(message);
    }

    public static void main (String args[])
    {
        HelloWorld afficheur = new HelloWorld();
        afficheur.setMessage ("Je suis un objet");
        afficheur.afficheMessage();
    }
}
```

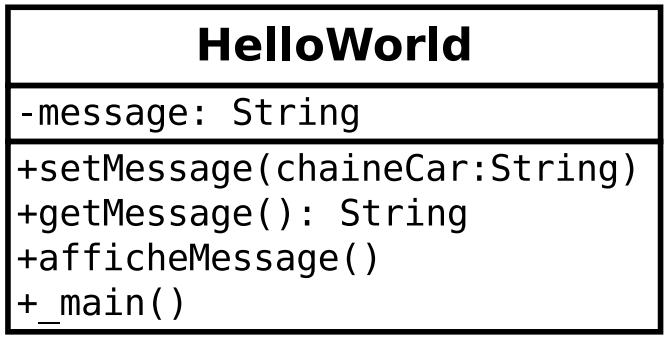
Hello world, une seule classe

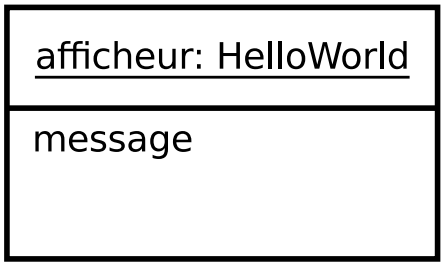
```
public class HelloWorld
{
    private String message;

    public void setMessage (String chaineCar)
    {
        message = chaineCar;
    }

    public String getMessage ()
    {
        return message;
    }

    public void afficheMessage ()
    {
        System.out.println(message);
    }

    public static void main (String args[])
    {
        HelloWorld afficheur = new HelloWorld();
        afficheur.setMessage ("Je suis un objet");
        afficheur.afficheMessage();
    }
}
```

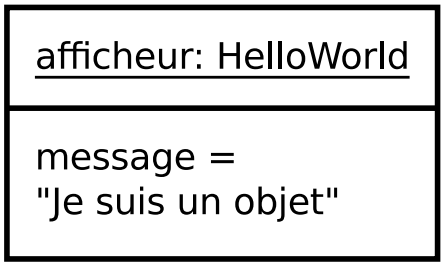
Hello world, une seule classe

```
public class HelloWorld
{
    private String message;

    public void setMessage (String chaineCar)
    {
        message = chaineCar;
    }

    public String getMessage ()
    {
        return message;
    }

    public void afficheMessage ()
    {
        System.out.println(message);
    }

    public static void main (String args[])
    {
        HelloWorld afficheur = new HelloWorld();
        afficheur.setMessage ("Je suis un objet");
        afficheur.afficheMessage();
    }
}
```



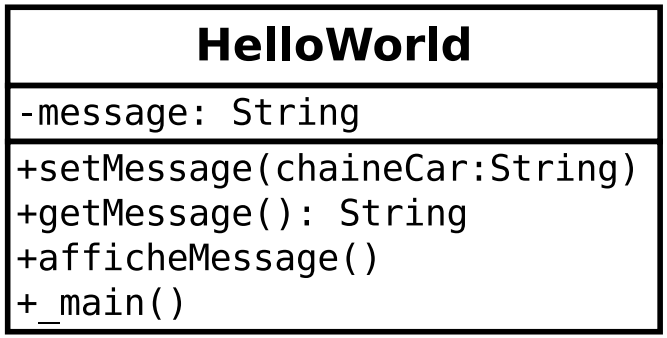


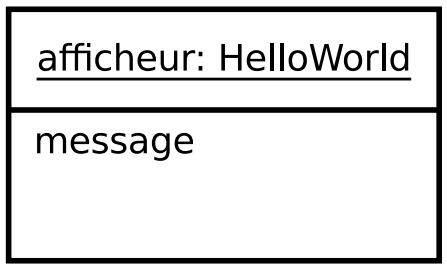


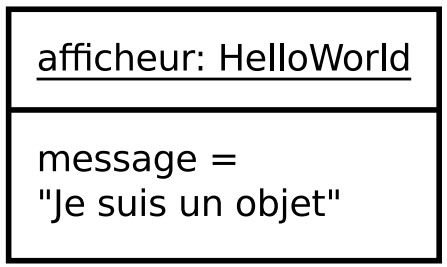


Hello world, une seule classe

```
public class HelloWorld
{
    private String message;

    public void setMessage (String chaineCar)
    {
        message = chaineCar;
    }

    public String getMessage ()
    {
        return message;
    }

    public void afficheMessage ()
    {
        System.out.println(message);
    }

    public static void main (String args[])
    {
        HelloWorld afficheur = new HelloWorld();
        afficheur.setMessage ("Je suis un objet");
        afficheur.afficheMessage();
    }
}
```







Sortie :
Je suis un objet