Ex 1

1- On crée un fichier Test.java qui contient seulement :

```
public class Test
```

- Que se passe t'il si on cherche à le compiler ?

La définition de la classe Test est incomplète, pour définir une classe, il faut mettre deux accolades qui contiennent la définition de cette classe, quand bien même cette définition est vide :

```
public class Test
        {
        }
```

2- On crée une classe Test qui est la forme minimale que peut avoir une classe :

```
public class Test
        {
        }
```

L'ordre de compilation pour cette classe est javac Test.java, l'ordre d'exécution est java Test.

- La compilation fonctionne t'elle ?

La compilation fonctionne ici, on constate que dans le repertoire bin apparaît un fichier Test.class contenant les bytecodes qui résultent de la compilation de Test.java.

- Que se passe t'il lorsque on demande à la machine d'exécuter le programme Test en faisant java Test en ligne de commande ? Quelle message renvoie la machine ? Comment interprétez vous ce message ?

En appellant l'exécution du code, on obtient le message d'erreur suivant :

```
Exception in thread "main" java.lang.NoSuchMethodError: main
```

Lorsque l'on appelle java Test, on indique à la machine de lancer le programme à partir de Test. Pour ce faire la mahcine cherche la fonction **static void main(String[] args)** dans la définition de la classe Test. Elle ne s'y trouve pas : la machine renvoie donc un message d'erreur.

3- On modifie la classe Test :

```
public class Test{

        public static void main(String[] args)
        {
        }
}
```

à l'identique, on la compile (javac Test.java) puis on l'exécute : java Test.

- Que constatez vous A l'exécution ?

On constate qu'à l'exécution il ne se produit rien. Notamment, il n'y a plus de message d'erreur. En effet, comme désormais la définition de la classe Test contient la définition de la méthode/fonction **main(String[] args)**, il devient possible de lancer l'exécution de cette fonction lors de l'appel java Test. La machine exécute toutes les instructions de la fonction main (qui n'en contient aucune) et se termine.

- Peut-on envisager de faire un programme plus court que celui-ci ?

Non. Le minimum est de définir une classe et une fonction **static void main(String[] args)** dans cette classe.

4- On crée une classe Test qui est de la forme :

```
public class Test{

        public static void main(String[] args)
        {
        System.out.println("Ceci vient du  main de Test");
        }
}
```

- Que donne l'exécution de ce code ?

Ce code fonctionne et affiche **Ceci vient du main de Test** dans l'invite de commandes.

- Question complémentaire : que donnerait la compilation et l'exécution de ce programme :

```
public class Test{

        public static void main(int args)
        {
        System.out.println("Ceci vient du  main de Test");
        }
}
```

Ici, on défini une fonction **static void main(int args)** et non pas une fonction **static void main(String[] args)** : notre fonction prend en paramètre un objet de type int et non pas un tableau d'objets de type String ainsi que doit la faire la fonction main d'un programme. Une fonction est définie par un nom et une liste de paramètres. La fonction main a un seul paramètre qui est de type String[] (tableau d'objets String).

Ex 2

1- On veut exécuter le code suivant

```
int i;
System.out.println(i);
ActifNegotiable a;
System.out.println(a);
```

- Créez une classe Test contenant une fonction main qui exécute ces lignes de code. La compilation fonctionne t'elle ?

On crée une classe Test.java :

```
public class Test3{
        public static void main(String[] args)
        {
        int i;
        System.out.println(i);
        ActifNegotiable a;
        System.out.println(a);
        }
}
```

En essayant de compiler cette classe, on obtient une erreur :

```
Test.java:6: cannot find symbol
symbol  : class ActifNegotiable
location: class Test
    ActifNegotiable a;

1 error
```

Celle ci signifie que la compilation n'a pu se faire, à cause de la 6ème ligne du fichier. La machine indique qu'elle n'a pas trouvé le symbole ActifNegotiable : effectivement, à la 6ème ligne, on cherche à définir un objet a de type ActifNegotiable. Hors, aucune définition de cette classe n'est accessible pour la machine.

2- Corrigez l'erreur de la question précédente. On veut que ActifNegotiable soit une classe avec les attributs suivants :

```
double cours;
String nom;
```

- Créez cette classe de manière à ce qu'elle soit "vue" par la classe Test.java (il existe trois solutions, vous devez en connaître au moins deux à ce stade).

On crée la classe ActifNegotiable. Pour qu'elle soit vue par la classe Test.java, soit on ajoute sa définition à la fin du fichier Test.java, après la définition de la classe Test.java :

```
class ActifNegotiable
        {
        double cours;
        String nom;
        }
```

Soit, on crée un nouveau fichier : ActifNegotiable qui contient la définition de la classe ActifNegotiable et est placé dans le même répertoire que la classe Test :

```
public class ActifNegotiable
        {
        double cours;
        String nom;
        }
```

La troisième solution consisterait à mettre le fichier contenant la définition de la classe ActifNegotiable n'importe où sur le disque dur. Elle n'est pas implémentée ici.

3- Une fois la classe ActifNegotiable définie, on recompile la classe Test et on relance l'exécution des lignes de code de 1-

- La compilation provoque t'elle encore une erreur ?

On obtient le message d'erreur suivant :

```
Test.java:6: variable i might not have been initialized
      System.out.println(i);

Test.java:8: variable a might not have been initialized
      System.out.println(a);

2 errors
```

Il se produit que l'on appelle l'affichage du contenu de la variable i à la ligne 6, l'affichage du contenu de la variable a à la ligne 8 : les versions récentes du compilateur java interdisent en général la manipulation des variables qui n'ont pas reçu d'initialisation.

4- On modifie le code la fonction main :

```
int i=0;
System.out.println(i);
ActifNegotiable a=null;
System.out.println(a);
```

Que se passe t'il à l'exécution de ce code ?

- L'exécution ne provoque plus d'erreur. Comment comprenez vous ce qui apparaît à l'écran ?

A l'écran apparaît :

```
0
null
```

Le code affecte 0 à la variable i, c'est donc la valeur 0 qui est affichée avec l'instruction System.out.println(i);

Pour la variable objet, elle est initialisée avec le pointeur null : une variable objet a pour contenu l'adresse d'une variable objet en mémoire vive. Lorsqu'elle est à null, cela signifie qu'elle ne pointe sur aucune adresse. Pour une variable objet, elle peut être :

(1) non initialisée,

(2) initialisée à null,

(3) initialisée avec une adresse d'objet.

5- On rajoute une instruction :

```
int i=0;
System.out.println(i);
ActifNegotiable a=null;
System.out.println(a);
System.out.println(a.cours);
```

- Quel affichage constate t-on ? Qu'est ce qui vous paraît manquer ? Corrigez le code sans éliminer aucune instruction : vous avez juste le droit d'en modifier une. Expliquez précisement l'affichage qui s'en déduit.

On constate l'affichage suivant :

```
0
null
Exception in thread "main" java.lang.NullPointerException
	at Test.main(Test.java:9)
```

On tombe sur une Exception classique (une Exception est une erreur à l'exécution) : une NullPointerException, elle signifie qu'on cherche à faire des calculs ou des manipulations sur une variable objet qui est initialisée à null. En effet, ici, à la dernière ligne,

```
System.out.println(a.cours);
```

On cherche à afficher le cours de l'ActifNegotiable a, or la variable objet a a été initialisée à null : elle ne pointe sur l'adresse d'aucun objet, il est donc impossible de récupérer le cours associé à l'ActifNegotiable a.

Pour que le code fonctionne, il faut préalablement créer un objet de type ActifNegotiable et affecter son adresse à a. Comme on ne dispose par d'objet ActifNegotiable à ce point il convient de le créer. Pour créer une variable objet de type ActifNegotiable, on fait appel à son constructeur, la fonction qui va réserver l'espace nécessaire en mémoire pour stocker la variable objets et ses différents champs. En allant voir la classe ActifNegotiable que nous avons créée, nous constatons qu'elle ne contient aucun constructeur (elle ne contient aucune méthode et donc a fortiori aucun constructeur). Puisque la classe ActifNegotiable n'a pas de constructeur défini, on utilise le constructeur par défaut pour construire des objets. On modifie le main de manière appropriée :

```
int i=0;
System.out.println(i);
ActifNegotiable a=new ActifNegotiable();
System.out.println(a);
System.out.println(a.cours);
```

On constate un affichage de ce type à l'exécution :

```
0
ActifNegotiable@addbf1
0.0
```

On n'affecte plus le pointeur null à la variable objet a, mais un objet que l'on construit de manière explicite, c'est à dire pour lequel un réserve un espace mémoire dans lequel on stocke la valeur de ses attributs. La valeur contenue dans a est alors l'adresse de l'objet ActifNegotiable en mémoire. **ActifNegotiable@addbf1** signifie "l'objet de type ActifNegotiable à l'adresse addbf1". Le constructeur initialise implicitement les variables : il affecte 0 aux variables numériques et null aux variables objets. Notamment, a.cours est initialisée à 0.

6- On veut maintenant qu'il soit impossible de construire un objet ActifFinancier sans lui attribuer une valeur, de sorte que la valeur du cours ne soit pas 0 après la construction de l'objet.

On modifie la définition de la classe ActifNegotiable en définissant explicitement un constructeur qui prend un paramètre double :

```
public class ActifNegotiable
        {
        double cours;
        String nom;

        public ActifNegotiable(double x)
                {
                this.cours=x;
                }
        }
```

Le constructeur est la seule fonction-méthode qui ne spécifie pas de type de retour et porte le nom de la classe dans laquelle il se trouve. Ici, on crée un constructeur tel qu'il faut lui passer une valeur. Donc, à chaque fois qu'un objet ActifNegotiable sera construit, ce sera avec une valeur passée en entrée. Dans ce cas, le constructeur ActifNegocable(double x) commence par initialiser toutes les variables à 0 ou null puis il exécute l'instruction this.cours=x;

7- Que se passe t'il à l'exécution du code suivant :

```
int i=0;
System.out.println(i);
ActifNegotiable a;
a=new ActifNegotiable(5);
System.out.println(a);
System.out.println(a.cours);
ActifNegotiable a2=new ActifNegotiable();
```

Que se passe t'il à l'exécution de ce code ? Si finalement on veut pouvoir construire des ActifNegotiable en disposant de leur cours initial ou sans en disposer, comment faire ?

Dans ce cas, la compilation produit une erreur :

```
Test.java:11: cannot find symbol
symbol  : constructor ActifNegotiable()
location: class ActifNegotiable
ActifNegotiable a2=new ActifNegotiable();

1 error
```

En effet, le constructeur par défaut n'est plus utilisable ici, donc **new ActifNegotiable()** n'est pas possible comme instruction. Il est possible de réintroduire ce constructeur (auquel cas il sera possible de construire des objets ActifNegotiable sans passer de valeur en paramètre) :

```
public class ActifNegotiable
        {
        double cours;
        String nom;
```

```
    public ActifNegotiable(double x)
        {
        this.cours=x;
        }

    public ActifNegotiable()
        {
        }
}
```

Notez qu'il est possible de définir des instructions pour le constructeur sans paramètres une fois qu'on l'a défini explicitement.

Ex 3

0- Quelques élements préliminaires. String est une classe. Elle s'instancie par un constructeur qui prend une chaîne en entrée :

String str=new String("Coucou");

Pour cette classe, une autre forme d'instanciation est possible et équivalente :

String str="Coucou";

Quand on utilise **System.out.println("");**, on utilise en fait la fonction **System.out.println(String str);**. Donc

```
String str="Coucou";
System.out.println(str)
```

donne le même affichage que

System.out.println("Coucou");

1- Quelle différence faites-vous entre "c", 'c' et c ?

"c" est une chaîne de caractère qui contient 1 unique caractère c.

'c' est un char.

c est une variable qui peut être de type objet ou de type primitif.

2- Dans quelles conditions le test v=="c1" ne provoquera pas d'erreur à la compilation ? Qu'en est-il des tests v=='c1' et v==c1 ?

v=="c1" ne provoquera pas d'erreur si v est une variable de type String.

v=='c1' ne provoquera pas d'erreur si v est une variable de type char

v==c1 ne provoquera pas d'erreur si v et c1 sont deux variables de même type.

3- Soit le code suivant, qu'affiche t'il ? Pourquoi ?

```
char c='a';
char x='a';
boolean bo=(c==x);
System.out.println(bo);
String str=new String("President");
String str2=new String("President");
boolean b=(str==str2);
```

```
System.out.println(b);
String str3=str2;
boolean b1=(str3==str2);
System.out.println(b1);
boolean b2=(str2.equals(str));
System.out.println(b2);
```

Ce code code affiche

True
False
true
true

Pour évaluer c==x, la machine récupère le contenu stocké pour la variable c qui correspond au code de la valeur 'a', elle récupère le contenu stocké pour la variable x qui correspond au code de la valeur 'a' : les deux variables ont bien le même contenu et bo prend bien la valeur vraie.

Pour évaluer str==str2, la machine va chercher le contenu de la variable str : c'est l'adresse de l'objet crée à la ligne **String str="President";** , la machine va ensuite chercher le contenu de la variable str2 : c'est l'adresse de l'objet crée à la ligne **String str2="President";**, str et str2 correspondent à deux objets différents enregistrés à deux adresses différentes : str et str2 ne contiennent pas la meme adresse et str==str2 renvoie false.

Avec **String str3=str2;**, on affecte à str3 l'adresse de str2 et donc str3==str2 est vrai et donc l'affichage de b1 onne true.

Enfin, si st1==str2 ne permet pas de comparer les chaines str str2, il existe une fonction de String qui prend une String en paramètre et renvoie un booléen si les deux chaines str et str2 sont les mêmes : **str.equals(str2)** qui est absolument équivalente à **str2.equals(str)**. Si les deux objets sont différents, c'est bien la même chaine qui est stockée et la fonction renvoie vrai pour b2.

Dans le code qui précède combien d'objets String ont été crées ?

Deux objets String ont été créés (deux appels au constructeur) et un trois variables objets de type String ont été créées, avec deux variables objets qui pointent sur le même objet String.

4- Soit le code suivant, que produit-il ?

```
String str="";     int i=7;
str+='P';          str+="r";          str+='e';str+="s";          str+="i";          str+="d";
str+='e';          str+="n";          str+='t'; str+=i;
System.out.println(str);
```

Le code précédent crée un objet String qui est la chaine vide puis lui ajoute le caractère 'P', le caractère 'e', une chaine contenant "s" etc.... Au final, **President7** est affiché. Il est intéressant de voir qu'on peut faire évoluer une chaine en lui concaténant d'autres chaines, des char ou des int qui sont alors interprétés comme des chaines.

Quelle différence faites vous entre String str1=""; et String str2=" ";

str1 est la chaîne vide, str2 est la chaine qui contient 1 caractère espace. La nuance est la même que celle qui distingue l'ensemble vide : {} et l'ensemble contenant seulement 0 : {0}

5- Concaténation de deux String :

```
String str1="Les sanglots longs ";
String str2="des violons de l'automne";
String str3=str1+str2;
boolean b1=(str3==str1);
boolean b2=(str3==str2);
System.out.println(b1+" "+b2+" "+str3);
```

Quel affichage produit ce code ?

Ce code affiche **false false Les sanglots longs des violons de l'automne**. La chaine affichée est produite en affichant le contenu de la variable b1 en le transcivant comme une chaine, en le concaténant evec un espace vide, puis en concaténant le contenu de b2, puis de str3.

6- Quelle différence faites vous entre "2.55" et 2.55 ? Comment passer de l'un à l'autre ?

"2.55" est une chaine de caractères alors que 2.55 est une valeur numérique.

Pour obtenir une chaîne depuis une valeur numérique, doit double d cette valeur numérique :

```
double d=2.55;
String str="";
str=str+d;
```

Après ce code, str contient la chaîne "2.55".

Pour obtenir une valeur numérique depuis une chaine :

```
double d=0;
int i=0;
String str="2.24";
String str2="5";
        try
        {
        d=Double.parseDouble(str);
        i=Integer.parseInt(str2);
        }catch(Exception e){System.out.println("Erreur dans la recuperation de valeur numérique");}
System.out.println(d+" "+i);
```

Ce bout de code sera expliqué plus tard, il permet de récupérer dans les variables d et i les valeurs des chaines str et str2.

7- Création d'une classe de chaines URL, soit une classe de chaine contenant seulement des String commençant par "http://". A la construction des objets, de cette classe, il faut s'assurer de bien avoir une telle chaine.

Plusieurs solutions sont envisageables pour résoudre ce problème, on ne peut pas choisir entre elles ici, il faudra voir en fonction du contexte. Dans le cas où on sait que c'est un utilisateur humain qui rentre une chaine de caractères, dans ce cas, on peut lui passer un message en lui demandant de ressaisir une URL si la chaine passée ne commence pas par "http://". Dans ce cas,on lui met un message à l'écran en lui demandant de rentrer à nouveau une valeur :

```
 public class URL{
public String str;


class URL{
        public URL(String chaine)
        {
        char[] c=chaine.toCharArray(); // on crée un tableau de char à partir de la chaine.
                while(c.length<8 ||c[0]!='h'  ||c[1]!='t' ||c[2]!='t' ||c[3]!='p' ||c[4]!=':' ||c[5]!='/' ||c[6]!='/')
                {
                System.out.println("Votre url doit commencer par http://");
                chaine=Lire.S();
                c=chaine.toCharArray();
                }
        this.str=chaine;
        }
}
```

Dans le cas où on traite de données massives de manière automatique, il faut retenir un traitement automatique, un utilisateur ne peut pas modifier lui même des milliers, voire millions d'URL, une première solution serait alors :

```
 public class URL{
public String str;


class URL{
        public URL(String chaine)
        {
        char[] c=chaine.toCharArray(); // on crée un tableau de char à partir de la chaine.
                if(c.length<8 ||c[0]!='h'  ||c[1]!='t' ||c[2]!='t' ||c[3]!='p' ||c[4]!=':' ||c[5]!='/' ||c[6]!='/')
                {
                chaine="http://"+chaine;
                }
        this.str=chaine;
        }
}
```

On pourrait envisager beaucoup d'autres solutions, par exemple de tester automatiquement si l'URL est valable en essayant de s'y connecter et écarter celles des URL qui ne permettent pas une connection effective.