


# Feuille de TD 2

## Représentation mémoire

On dispose d'une classe "CompteBancaire" suivante :

 CompteBancaire
-proprietaire: String -solde: double
+CompteBancaire(nom: String) +CompteBancaire(nom: String, depotInitial: double) +getSolde(): double +debite(double) : void

1. Représentez la mémoire suite à l'exécution du code suivant :

```
public class Executable {  
    public static void main(String [] args) {  
        CompteBancaire riche = new CompteBancaire("Elon Musk", 19 000 000 000);  
        riche2 = riche;  
        riche.debite(1000 000);  
        System.out.println(riche2.getSolde());  
    }  
}
```

2. Représentez la mémoire à la suite de l'exécution du code suivant :

```
public class Executable {  
    public static void main(String [] args) {  
        ArrayList<String> liste = new ArrayList<>();  
        liste.add("Bonjour");  
        liste.add("Bonjour");  
        liste.add("Salut");  
        String chaine = "Bonjour";  
        String chaine2 = new String("Bonjour");  
    }  
}
```

3. Quels seraient les affichages si on ajoutait le code suivant ?

```
System.out.println(liste.get(0));  
System.out.println(liste.get(2));  
System.out.println(liste.get(0) == liste.get(1));  
System.out.println(liste.get(0) == liste.get(2));  
  
System.out.println(chaine == chaine2);  
System.out.println(chaine == liste.get(0));  
System.out.println(chaine2 == liste.get(0));
```

# Représentation mémoire avec appel de méthodes

On a la classe suivante :

```
public class Anniversaire{
    private ArrayList<String> invites;

    public Anniversaire(){
        this.invites = new ArrayList<>();
    }

    public void ajouteInvite(String nouvel_invite){
        boolean estDejaInvite = false;
        for(String invite: this.invites){
            if(invite.equals(nouvel_invite)){
                estDejaInvite = true;
            }
        }
        if(! estDejaInvite) // (3)
            this.invites.add(nouvel_invite);
    }
}
```

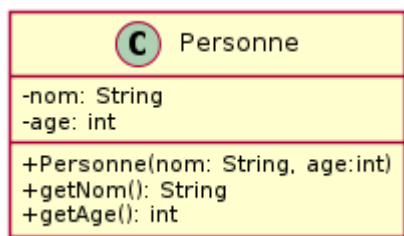
Et l'exécutable :

```
public class Executable {
    public static void main(String [] args) {
        Anniversaire a; // (1)
        a = new Anniversaire(); // (2)
        String john = "John";
        a.ajouteInvite(john);
    }
}
```

1. Indiquez l'état de la mémoire lorsqu'on passe pour la première fois par les lignes "(1)", "(2)" et "(3)".

## Algorithmes sur les listes

On a la classe suivante :



ainsi qu'une classe `Table` qui représentent l'ordre dans lesquels des convives sont placés à une table :

```

public class Table{
    private ArrayList<Personne> convives;

    public Table(){
        this.convives = new ArrayList<>();
    }

    public void ajouteConvive(Personne convive){
        this.convives.add(convive);
    }
}

```

1. Représentez la mémoire à la suite de l'exécution du code suivant :

```

public class Executable {
    public static void main(String [] args) {
        Table table = new Table()
        table.ajouteConvive(new Personne("Papa", 25));
        table.ajouteConvive(new Personne("GrandPapy", 85));
        table.ajouteConvive(new Personne("Mamie", 63));
        table.ajouteConvive(new Personne("Junior", 2));
        table.ajouteConvive(new Personne("Maman", 22));
        table.ajouteConvive(new Personne("Tonton", 47));
    }
}

```

- ajoutez à la classe Table une méthode 'sontACote(String nomPers1, String nomPers2)' permettant de savoir si deux personnes sont l'une à coté de l'autre.
- ajoutez une méthode `echange(String nomPers1, String nomPers2)` qui permet d'échanger le placement de deux personnes.
- ajoutez une méthode `doyen` qui renvoie l'âge de la personne la plus âgée de la table.
- ajoutez une méthode `estTrie` renvoyant true si la table est triée par ordre d'âge croissant.
- Bonus: ajoutez une méthode `trie` permettant de trier une table par age croissant.