

Cours6

Types abstraits – collections dynamiques

PLAN

- Type abstrait de données - encapsulation :
 - L'approche classique
 - L'approche objet / type abstrait
 - L'approche objet / utilisation
- Type abstrait de données : la collection dynamique *ArrayList*
 - La collection statique
 - La collection dynamique
- Type abstrait de données : le type *Duree*
 - Le type Duree : définition
 - Le type Duree : utilisation

Type abstrait de données - encapsulation

L'approche classique

- Les données globales et les traitements sont accessibles par tous les modules (les données ne sont pas « protégées ») : c'est le cas avec le Java R1.01.

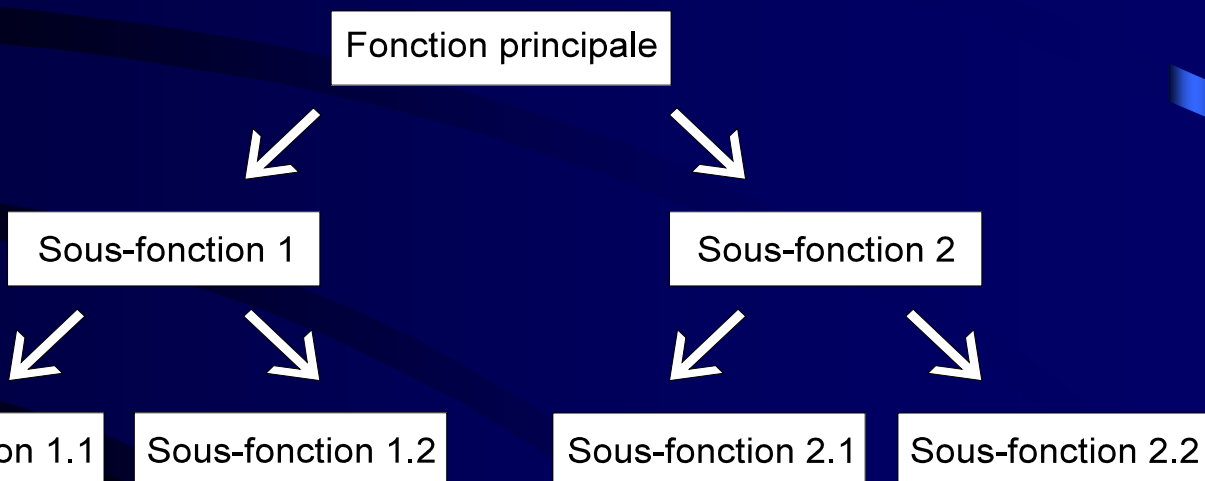
=> Il n'y a pas forcément de cohérence entre données et traitements. Exemple : un cercle ne peut pas avoir un rayon négatif.

- Un programme = un enchaînement de procédures et de fonctions

=> Exprimer le monde réel dans ces conditions peut être difficile.

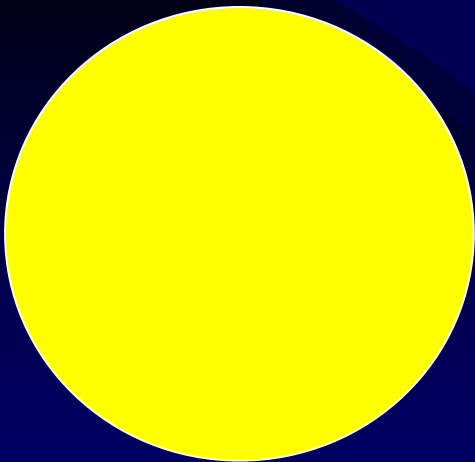
L'approche classique

Développement basé sur une décomposition algorithmique.



L'approche objet / type abstrait

- Les objets du monde réel nous entourent, ils naissent, vivent et meurent.
- En informatique, l'approche **objet**, c'est modéliser l'objet réel avec une version simplifiée. Un objet contient des informations qui le caractérisent.
- Exemple :



```
class Cercle {  
  
    int rayon;  
    float x, y;  
  
}
```

L'approche objet / type abstrait

- Autre exemple :



```
class Etudiant {  
  
    String nom;  
    int age;  
  
}
```

- Les objets sont des abstractions :
 - Une abstraction est un résumé, un condensé
 - Mise en avant des caractéristiques essentielles
 - Dissimulation des détails
 - Une abstraction se définit par rapport à un point de vue

L'approche objet / type abstrait

- Les objets (réels ou modélisés) ont des comportements. Ces comportements sont modélisés par des fonctions (méthodes).
- Exemple

```
class Cercle {  
  
    int rayon;  
    float x, y;  
  
    void changerRayon(...) {...}  
  
    void deplacerCercle (...) {...}  
  
}
```

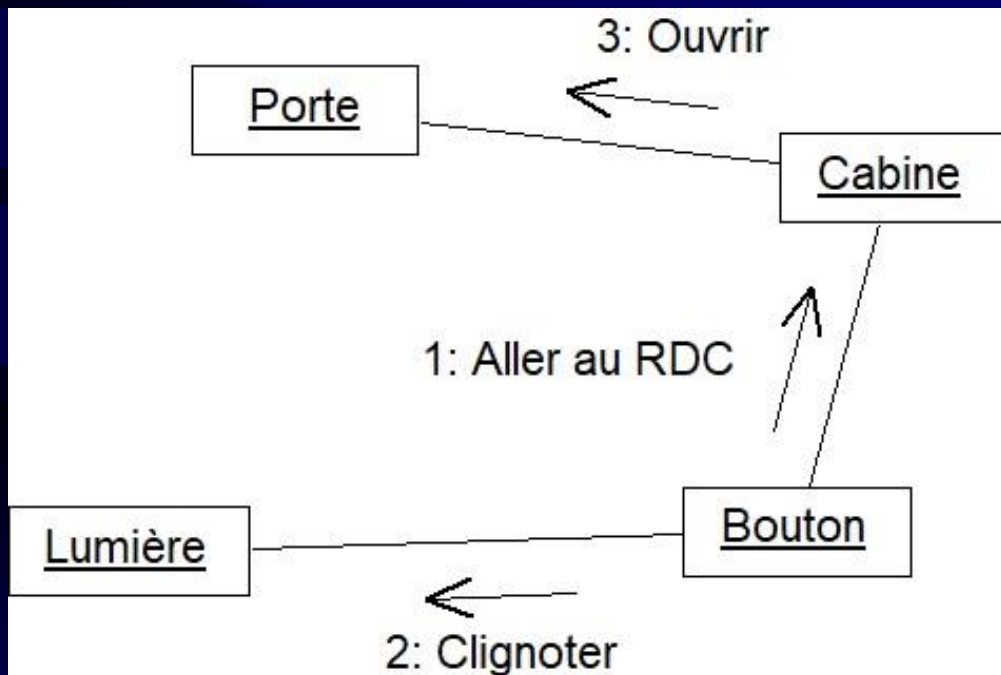

L'approche objet / type abstrait

- Les données (informations) et les comportements d'un objet doivent être cohérents = **encapsulation**.
- Exemple

```
class Cercle {  
    int rayon;  
    float x, y;  
  
    void changerRayon(int newR) {  
        if ( newR <= 0 ) {  
            Sop ("Erreur...");  
            ...  
        }  
    }  
    ...  
}
```

L'approche objet / type abstrait

- Les objets d'une application (informatique) collaborent entre eux par l'intermédiaire de messages.
- Exemple



L'approche objet / utilisation

- Le **Cercle** s'utilise en créant d'abord une instance

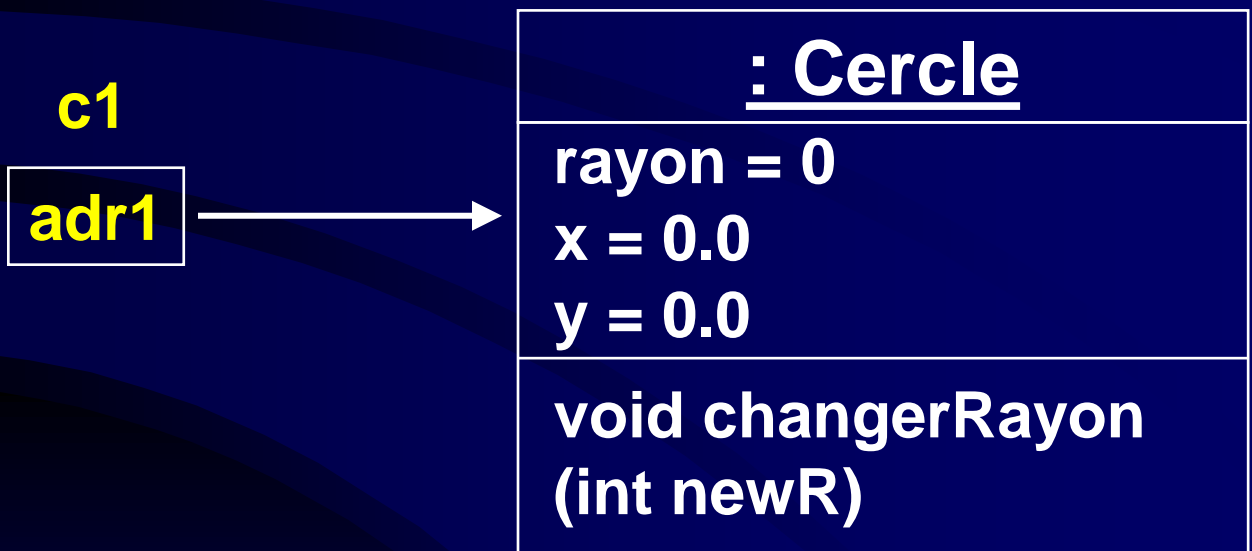
```
Cercle c1; // variable c1  
c1 = new Cercle(); // création
```

- ENSUITE, on peut utiliser les méthodes (fonctions)

```
c1.changerRayon ( 4 );
```

L'approche objet / utilisation

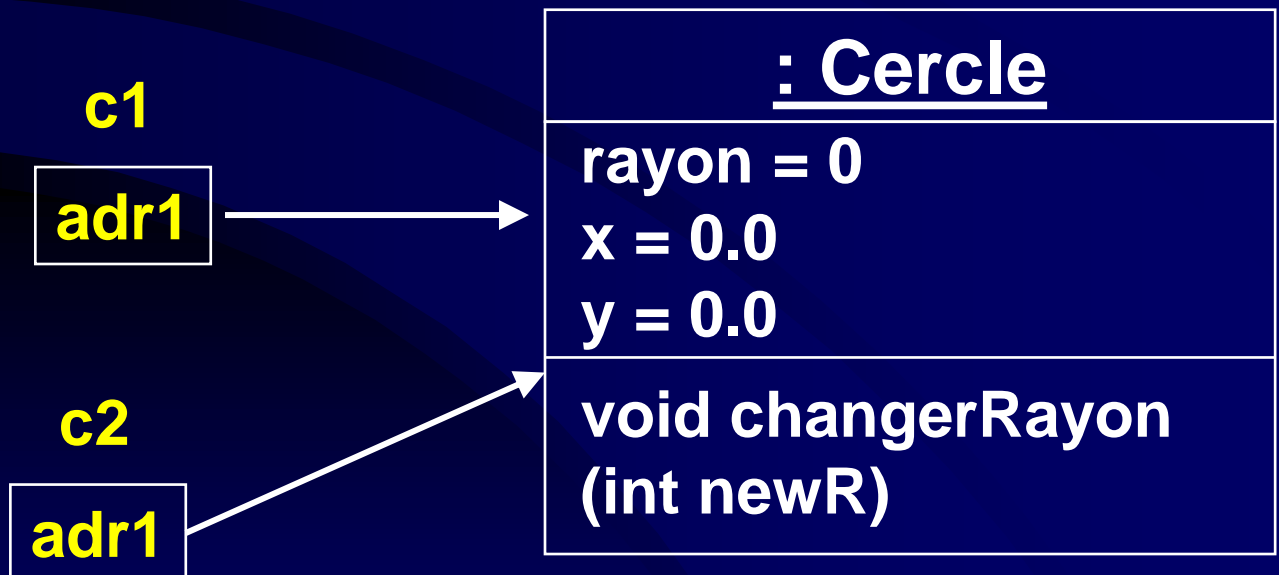
Attention, **c1** n'est pas un objet mais une variable contenant l'adresse de l'objet.



L'approche objet / utilisation

Et donc l'affectation entre variables de type objet ne correspond PAS à une recopie d'objet !!

```
Cercle c1;           // variable c1  
Cercle c2 = null;    // variable c2  
c1 = new Cercle();   // création  
c2 = c1;             // affectation
```



L'approche objet / utilisation

Et donc la comparaison entre variables de type objet correspond à une comparaison d'adresses !!

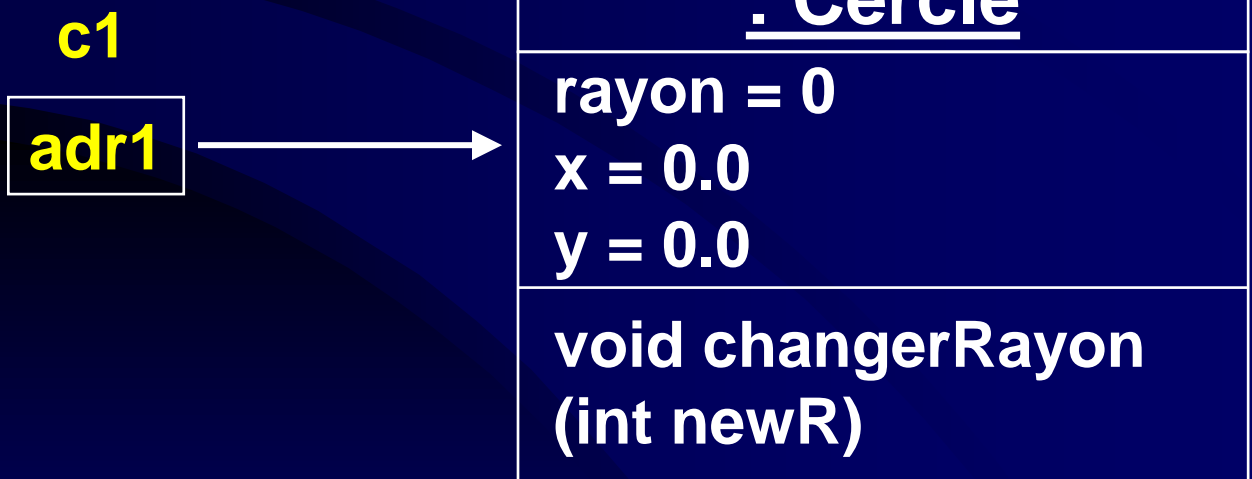
```
Cercle c1;      // variable c1
Cercle c2 = null; // variable c2
c1 = new Cercle(); // création

// compilation OK
// exécution OK
// évaluation condition : faux
if ( c1 == c2 ) {
    ...
    ...
}
```

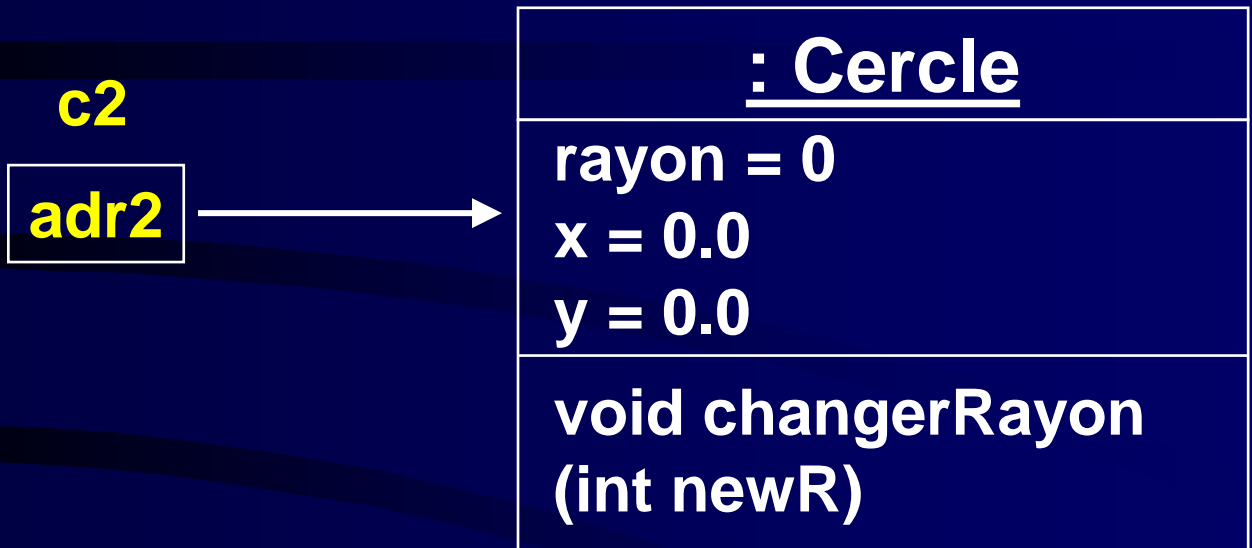
L'approche objet / utilisation

Si on veut comparer le contenu des objets (i.e. les données) il faut utiliser la méthode equals() !!

```
Cercle c1, c2; // variables c1, c2  
c2 = new Cercle(); // création  
c1 = new Cercle(); // création
```



L'approche objet / utilisation



```
// compilation OK
// execution OK
// évaluation condition : vrai
if ( c1.equals(c2) ) {
    ...
    ...
}
```


Type abstrait de données - la collection ArrayList

La collection statique

La collection statique = le tableau

```
int[] tab;    // null par défaut  
tab = new int [4]; // 4 cases
```



La collection statique



Inconvénients :

- le nombre de cases est figé
- la suppression d'une case n'est pas possible
- le redimensionnement du tableau n'est pas possible
- la duplication du tableau nécessite un nouveau tableau avec une recopie case par case

La collection dynamique

Exemple de collection dynamique = type abstrait **ArrayList** de Java

Avantages :

- à la création, il n'y pas encore de collection (size = 0)

```
ArrayList tab = new ArrayList();
```

- on DOIT préciser le type de données

```
ArrayList<Integer> tab = new  
ArrayList<Integer>();
```

- l'ajout d'un élément entraîne la création d'une case en plus

```
tab.add ( 45 ); // size = 1
```

La collection dynamique

Exemple de collection dynamique = type abstrait **ArrayList** de Java

Avantages :

- on peut enlever une donnée ce qui entraîne la destruction de la case

```
tab.remove ( 0 ); // size = 0
```

- on peut enlever toutes les données de la collection

```
tab.clear(); // size = 0
```

- on peut afficher TOUTE la collection en 2 lignes

```
String st = tab.toString();  
System.out.println ( st );
```

- etc.

Type abstrait de données - le type Duree

Le type Duree : définition

But : utiliser un type abstrait **Duree** pour gérer le temps (dans une application d'horaires de trains par exemple).

```
class Duree {  
    // en millisecondes  
    int leTemps;  
  
    // une durée se construit à  
    // partir d'un nbre d'heures,  
    // de minutes, de secondes  
    Duree(int heures, int  
minutes, int secondes) {...}  
  
    // on peut connaître le temps  
    // en millisecondes  
    int getLeTemps() {...}  
  
    ...  
}
```

Le type Duree : définition

```
class Duree {  
    // en millisecondes  
    int leTemps;  
  
    ...  
  
    // on peut la comparer à une  
    // autre Duree  
    int compareA(Duree autreD) {...}  
  
    // on peut lui ajouter une  
    // autre Duree  
    void ajoute(Duree autreD) {...}  
  
    // on peut obtenir le temps  
    // sous forme textuelle  
    String enTexte(char mode) {...}  
}
```


Le type Duree : utilisation

Comment utiliser le type **Duree**. Exemple.

```
void uneMeth () {  
  
    // création de la Duree « d1 »  
    // 2h / 40 min / 15 sec  
    Duree d1=new Duree(2, 40, 15);  
  
    // création de la Duree « d2 »  
    // 3h / 10 min / 25 sec  
    Duree d2=new Duree(3, 10, 25);  
  
    // ajout de d2 à d1  
    // attention c'est d1 qui est  
    // modifié et PAS d2  
    d1.ajoute ( d2 );  
  
    // affichage de d1 sous forme  
    // textuelle  
    String s1 = d1.enTexte ( 'H' );  
    System.out.println ( s1 );  
}
```