Implementing Aggregation and Composition Relationships in Java

Dr. REEMA AL-KAMHA

مقدمة

- **تذكرة:**
  تم في المحاضرات السابقة التأكيد على أن مفهوم الصف مماثل لمفهوم النوع، و بالتالي استخدام الصف لتعريف أنواع جديدة

- سندرس في هذه المحاضرة بعض العلاقات التي قد توجد بين الصفوف. نذكر منها:
  - علاقة Aggregation
  - علاقة Composition

Aggregation

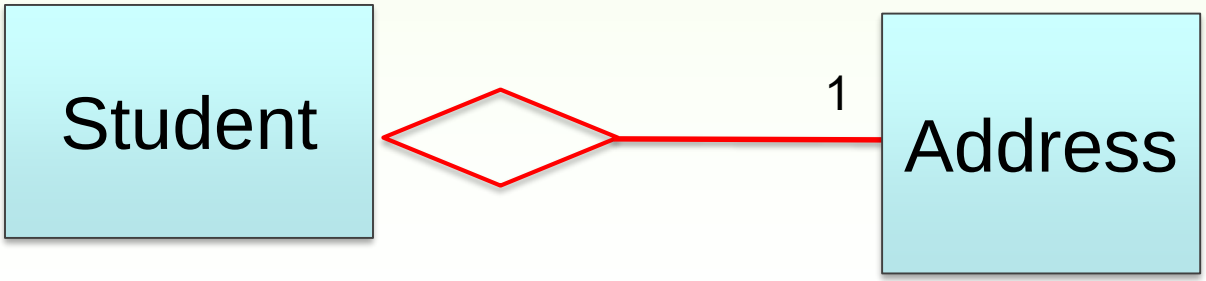
- It represents a **HAS-A** relationship.
- It is a **unidirectional association** i.e. a one way relationship. For example, department can have students but vice versa is not possible and thus unidirectional in nature.
- In Aggregation, **both the entries can survive individually** which means ending one entity will not effect the other entity

- تمثيل لعلاقة HAS-A يملك أو له أو يحوي
- هي علاقة باتجاه واحد. فمثلا يمكن للقسم أن يحوي طلابا و لكن العكس غير صحيح
- في علاقة الـ Aggregation وجود كل طرف من أطراف العلاقة غير متعلق بالطرف الآخر، فإنهاء أحدهما لن يؤثر على وجود الآخر

Aggregation

- It represents a **HAS-A** relationship.

- لتوضيح علاقة الـ Aggregation نذكر المثال التالي:
  - ليكن لدينا صفين Student و Address
  - كل طالب له (HAS-A) عنوان، و بالتالي العلاقة بين صف الطالب وصف العنوان هي علاقة Aggregation
  - ولكن من جهة أخرى، فإن صف العنوان هو صف مستقل غير متعلق بالضرورة بصف الطالب (أي غير معتمد بوجوده على وجود صف الطالب)، فهو صف موجود بغض النظر عن وجود صف الطالب.
  - وبالتالي، فقد يتم حذف طالب معين بينما عنوانه يبقى موجودا و قد يسند هذا العنوان لطالب آخر
  - ومنه: Student HAS-A Address هي علاقة Aggregation، تمثل بلغة UML كالتالي:



Aggregation

- لننظر في مثال آخر لتوضيح علاقة الـ Aggregation :
  - صف المحفظة Wallet و صف النقود Money.
  - العلاقة بين الصفين هي أن المحفظة تحوي نقود (Wallet has Money) و لكن النقود لا تحتاج إلى محفظة بالضرورة لوجودها، وبالتالي علاقة باتجاه واحد.
  - في هذه العلاقة كل من الطرفين يمكن أن يبقى قائما بحد ذاته بغض النظر عن الطرف الآخر . في مثالنا إذا لم يوجد صف المحفظة، فهذا لا يعني أن صف النقود لا يمكن أن يوجد.
  - ومنه: Wallet HAS-A Money هي علاقة Aggregation

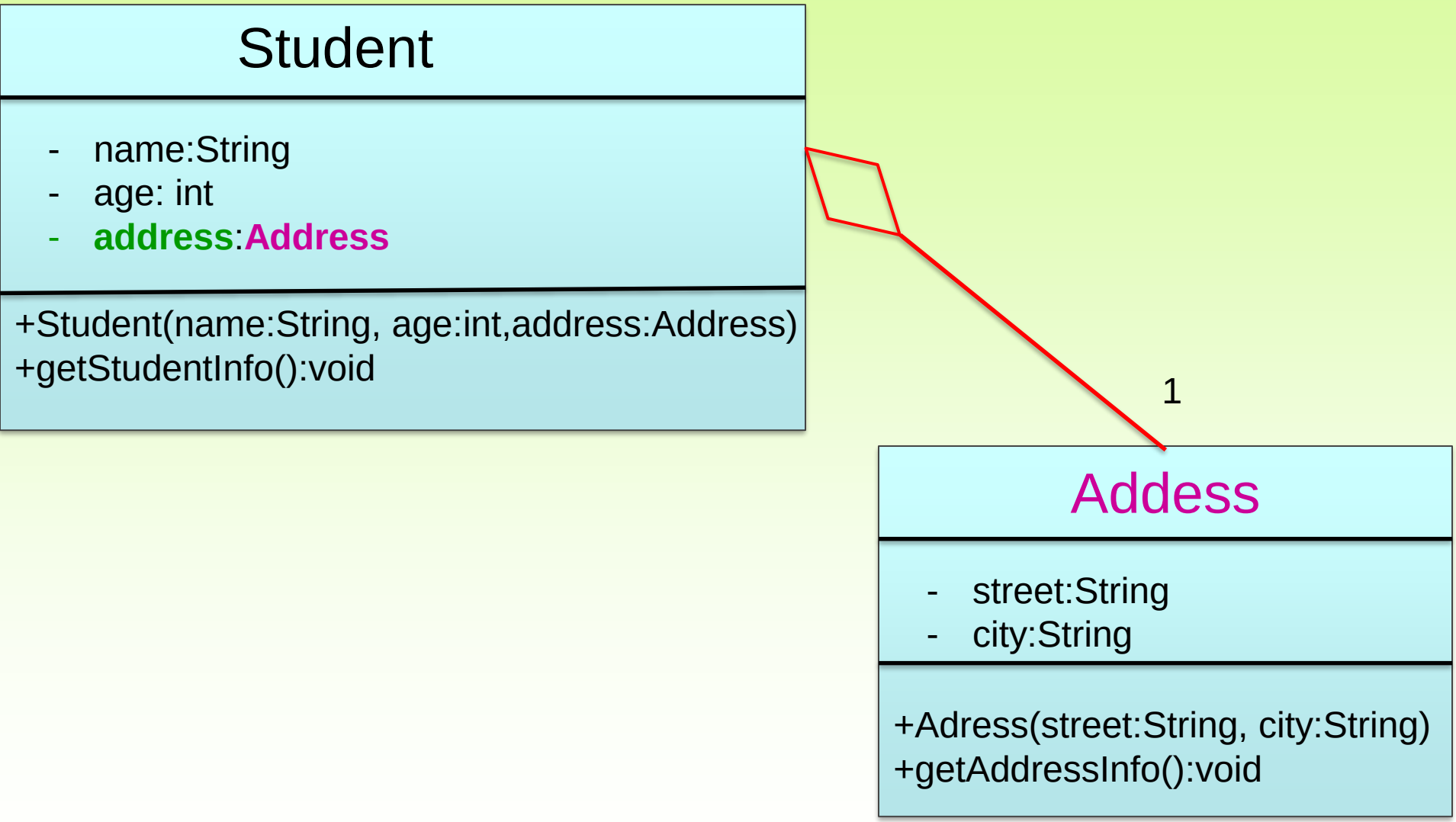
مثال على علاقة Aggregation في الجافا

اكتب برنامجا بلغة الجافا للتوصيف التالي:

- **تعريف صف اسمه Address** يحوي:
  - الحقول التالية: اسم الشارع street و اسم المدينة city
  - باني مناسب للتوصيف
  - طريقة getAddressInfo() والتي تطبع اسم الشارع و اسم المدينة
- **تعريف صف Student** يحوي:
  - الحقول التالية: اسم الطالب name، العمر age، و العنوانaddress (والذي هو من نوع الصف Address)
  - باني مناسب للتوصيف
  - طريقة getStudentInfo() والتي تطبع اسم معلومات الطالب
- **تعريف صف StudentAggregation** يقوم بإنشاء أغراض من الصف Student، و استدعاء الطرق الموجودة فيه

الحقل address هو من نوع الصف Address

تمثيل علاقة الـ Aggregation باستخدام لغة UML



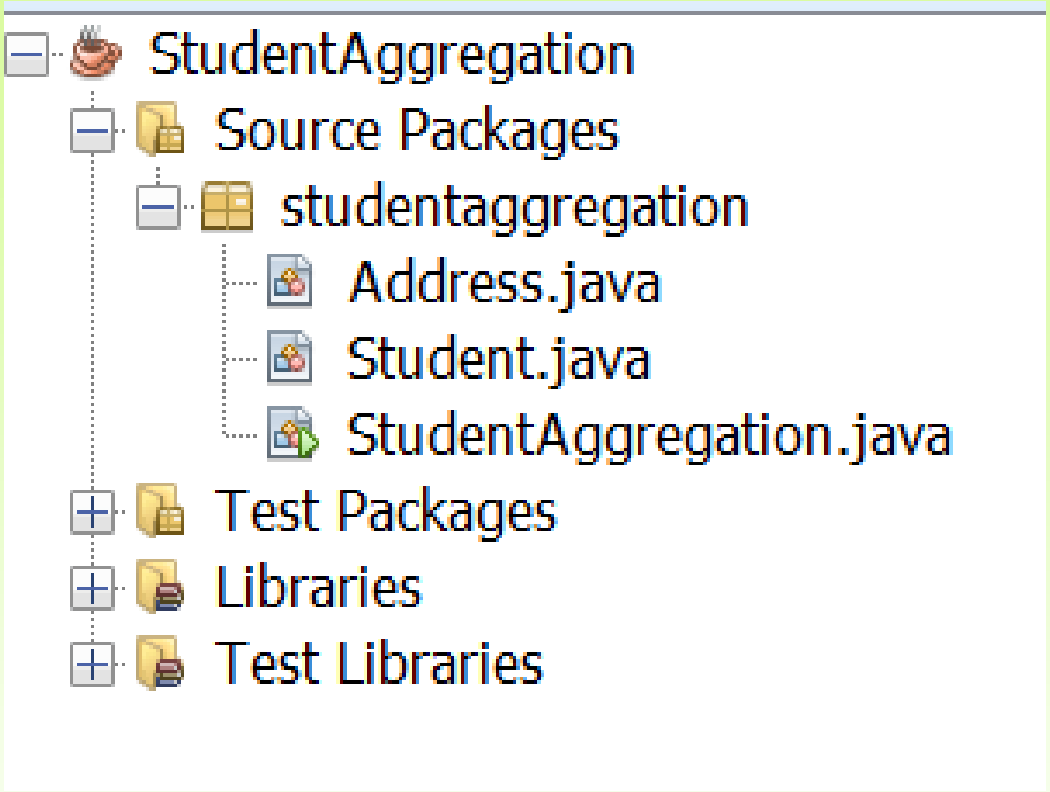
لتنفيذ علاقة Aggregation في الجافا، فإنه في الصف Student

- أولا:

يتم التصريح عن الحقل address في الصف Student على أنه من نوع الصف Address، و هذا الحقل حاليا خالي لا يحمل أي عنوان (أي لا يؤشر على أي غرض) من الصف Address

- ثانيا:

ضمن باني الصف Student يتم جعل الحقل address يؤشر على أغراض من الصف Address


StudentAggregation
Source Packages
studentaggregation
Address.java
Student.java
StudentAggregation.java
Test Packages
Libraries
Test Libraries

Address الصف

```
package studentaggregation;
public class Address {
  private String street;
  private String city;

  public Address(String street, String city){
    this.street=street;
    this.city=city;
  }

  public void getAddressInfo(){
   System.out.println("The Address \n"+"Street is "+street+" City is "+city);
  }
}
```

الصف Student

```
package studentaggregation;
public class Student {
  private String name;
  private int age;
//Creating HAS-A relationship with Address class
  private Address address;
  public Student(String name, int age, Address address){
    this.name=name;
    this.age=age;
          Address الحقل address للصف Student يؤشر فعليا على (يحمل عنوان) غرض من الصف
    this.address=address;
  }
   public void getStudentInfo(){
    System.out.println("Student Information\n"+"Name is "+name+
    " Age is "+age);
    address.getAddressInfo();
    System.out.println("************************************");
   }
}
```

لاحظ أنه تم استخدام address من نوع الصف Address كحقل في الصف Student

StudentAggregation الصف

```
package studentaggregation;
public class StudentAggregation {
  public static void main(String[] args) {
        //  Address إنشاء غرضين من الصف
    Address add1=new Address("Baramka", "Damascus");
    Address add2=new Address("Mezza","Damascus");
// you should pass add1 object which contains address which you can access in Student class
        //  Student إنشاء ثلاثة أغراض من الصف
    Student s1=new Student("Sami",20,add1);
    Student s2=new Student("Rami",21,add2);
    Student s3=new Student("Hani",19,add1);
        //  طباعة معلومات عن الأغراض المنشأة
    s1.getStudentInfo();
    s2.getStudentInfo();
    s3.getStudentInfo();
  }
}
```

التنفيذ

```
Student Information
Name is Sami Age is 20
The Address
Street is Baramka City is Damascus
**********************************
Student Information
Name is Rami Age is 21
The Address
Street is Mezza City is Damascus
**********************************
Student Information
Name is Hani Age is 19
The Address
Street is Baramka City is Damascus
**********************************
```

ملاحظات حول الصف Student

- السطر التالي

```
private Address address;
```

يعني التصريح عن الحقل address في الصف Student على أنه من نوع الصف Address، و هذا الحقل حاليا خالي لا يحمل أي عنوان (أي لا يؤشر على أي غرض) من الصف Address

- السطر this.address=address; في الباني التالي للصف Student

```
public Student(String name, int age, Address address){
    this.name=name;
    this.age=age;
    this.address=address;
```

يعني أنه عندما يتم إنشاء غرض (طالب) من الصف Student فإن الحقل address لهذا الغرض سيؤشر على (سيحمل عنوان) الغرض (address) من الصف Address

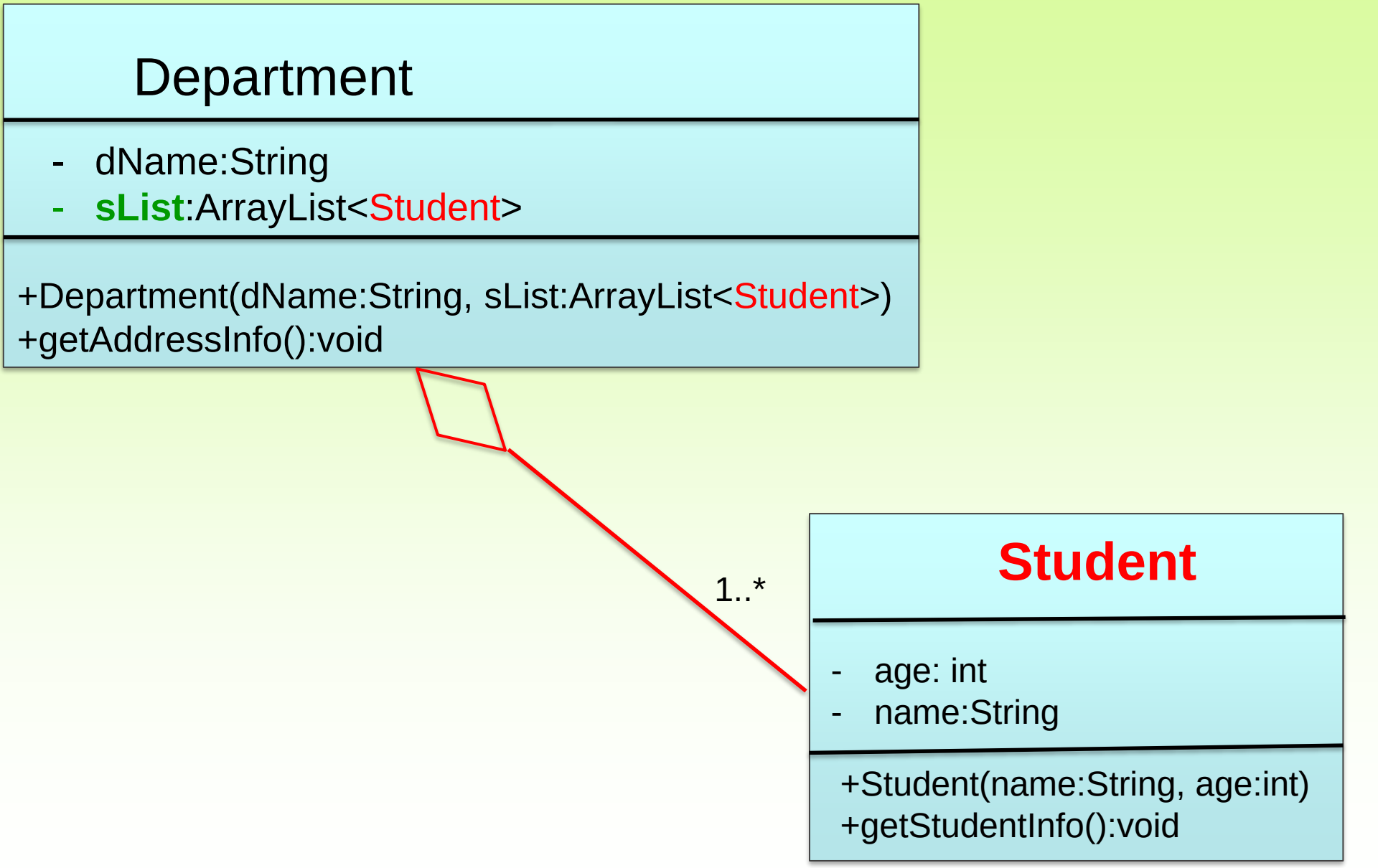
لاحظ أنه في حال حذف أي غرض من نوع Student فإن هذا لن يؤثر على الأغراض من نوع Address

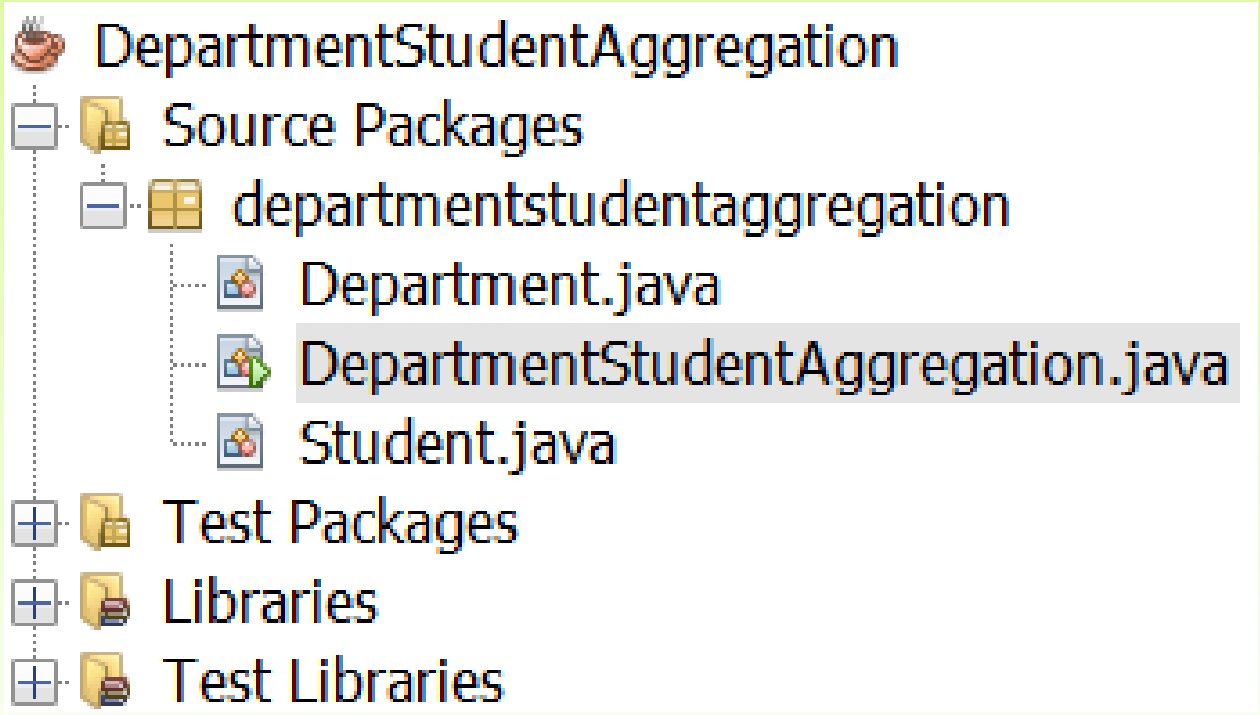
مثال آخر على علاقة Aggregation في الجافا

اكتب برنامجا بلغة الجافا للتوصيف التالي:

- **تعريف صف اسمه Student** يحوي:
  - الحقول التالية: اسم الطالب name و عمره age
  - باني مناسب للتوصيف
  - طريقة getStudentInfo() والتي تطبع معلومات الطالب
- **تعريف صف Department** يحوي:
  - الحقول التالية: اسم القسم dName، و **sList مجموعة من الطلاب من نوع الصف Student**
  - باني مناسب للتوصيف
  - طريقة getDeptInfo() والتي تطبع معلومات كل الطلاب في القسم
- **تعريف صف DepartmentStudentAggregation** يقوم بإنشاء أغراض من الصف Department، و استدعاء الطرق الموجودة فيه

تمثيل علاقة الـ Aggregation باستخدام لغة UML




DepartmentStudentAggregation
Source Packages
departmentstudentaggregation
Department.java
DepartmentStudentAggregation.java
Student.java
Test Packages
Libraries
Test Libraries

الصف Student

```
package departmentstudentaggregation;
public class Student {
  private String name;
  private int age;
  public Student(String name, int age){
    this.name=name;
    this.age=age;
  }
  public void getStudentInfo(){
    System.out.println("Student Information\n"+
                       "Name is "+name+" Age is "+age);
    System.out.println("*****************************");
  }
}
```

الصف Department

```
package departmentstudentaggregation;
import java.util.*;
public class Department {
 private String dName;
 private ArrayList<Student> sList;//ArrayList for all students
 public Department(String dName, ArrayList<Student> sList){
    this.dName=dName;
    this.sList=sList;
 }
 public void getDeptInfo(){
    System.out.println("Students who attend "+dName+" department");
    for(int i=0; i<sList.size();i++){
        sList.get(i).getStudentInfo();
    }
 }
}
```

ملاحظات:

- الحقل sList من نوع قائمة متجهية ArrayList و التي كل عنصر من عناصرها من نوع الصف Student
- sList.get(i) ترجع العنصر ذو الدليل i (والذي هنا غرض من الصف Student) من القائمة المتجهية

DepartmentStudentAggregation الصف

```
package departmentstudentaggregation;
import java.util.*;
public class DepartmentStudentAggregation {
    public static void main(String[] args) {
     Student s1=new Student("Ali", 20);
     Student s2=new Student("Sami", 21);
     ArrayList<Student> sList=new  ArrayList<Student> ();
     sList.add(s1);
     sList.add(s2);
     Department d1=new Department("CS",sList);
     d1.getDeptInfo();
    }
}
```

إضافة الغرض s1 إلى القائمة المتجهية

ملاحظات:

- حجز مكان في الذاكرة Heap لقائمة متجهية ArrayList كل عنصر من عناصرها من نوع الصف Student
- sList متغير مرجعي يتم حجز مكان له في الـ Stack يؤشر على القائمة المتجهية التي تم إنشاؤها في الـ Heap

التنفيذ

Students who attend CS department

Student Information

Name is Ali Age is 20

****

Student Information

Name is Sami Age is 21

****

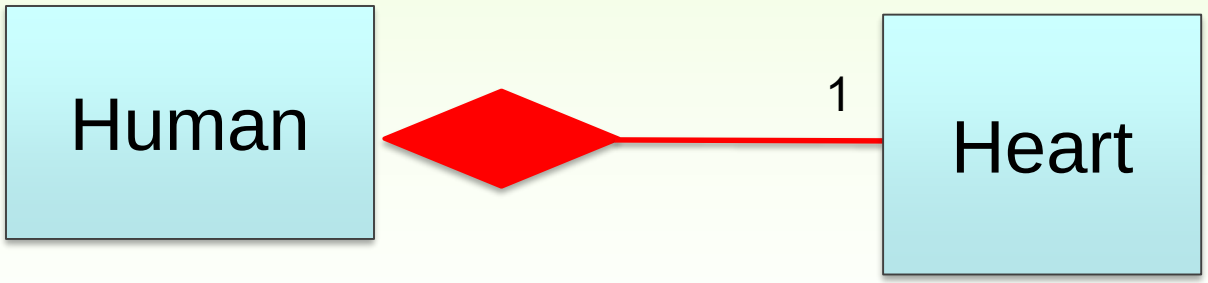
Composition

- It represents **part-of** relationship.
- In composition, both the entities are dependent on each other.
- When there is a composition between two entities, the composed object **cannot exist** without the other entity.

- علاقة الـ Composition هي نوع خاص من علاقة الـ Aggregation و تمثل علاقة جزء من حيث يعتمد فيها كلا الصفين على الآخر.
- عندما توجد علاقة Composition بين طرفين، فلا يمكن لأحدهما أن يكون موجودا مستقلا عن الطرف الآخر

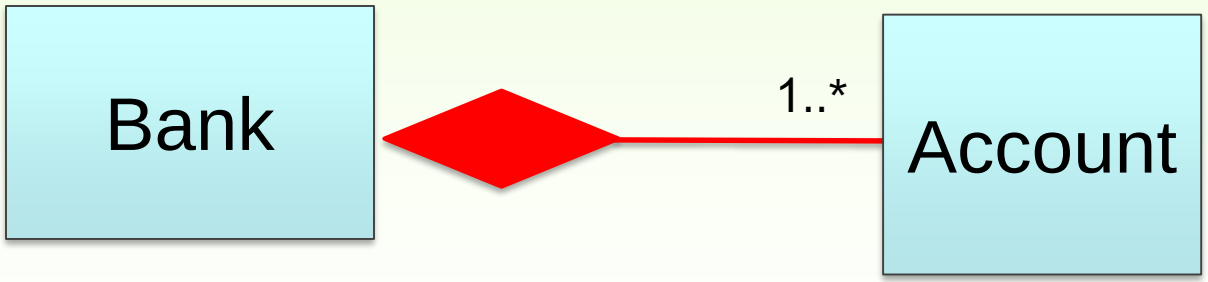
Composition

- فمثلا في صف الإنسان Human و صف القلب Heart. فإن القلب هو جزء من الإنسان. يحتاج الإنسان إلى القلب ليعيش و كذلك يحتاج القلب لأن يوجد في جسم الإنسان ليعمل.
- عندما يعتمد كلا الصفين على بعضهما و لهما نفس دورة الحياة (فعندما يموت أحدهما يموت الآخر أيضا)، وبالتالي تكون العلاقة Composition. فلا يوجد لصف القلب معنى في حال عدم وجود صف الإنسان.
- ومنه العلاقة بين: Human و Heart هي علاقة Composition، تمثل بلغة UML كالتالي:



Composition

- فمثلا في صف البنك Bank و صف الحساب Account. فإن البنك يتكون من قائمة الحسابات البنكية الموجودة فيه.
- بالتالي لا يمكن للبنك أن يوجد بدون وجود حسابات ، و كذلك لا يوجد للحسابات أي معنى بدون وجود البنك.
- عند حذف البنك فإن الحسابات البنكية سيتم حذفها، لأنه لا يوجد لها معنى في حال عدم وجود صف البنك..
- ومنه العلاقة بين: Bank و Account هي علاقة Composition، تمثل بلغة UML كالتالي:

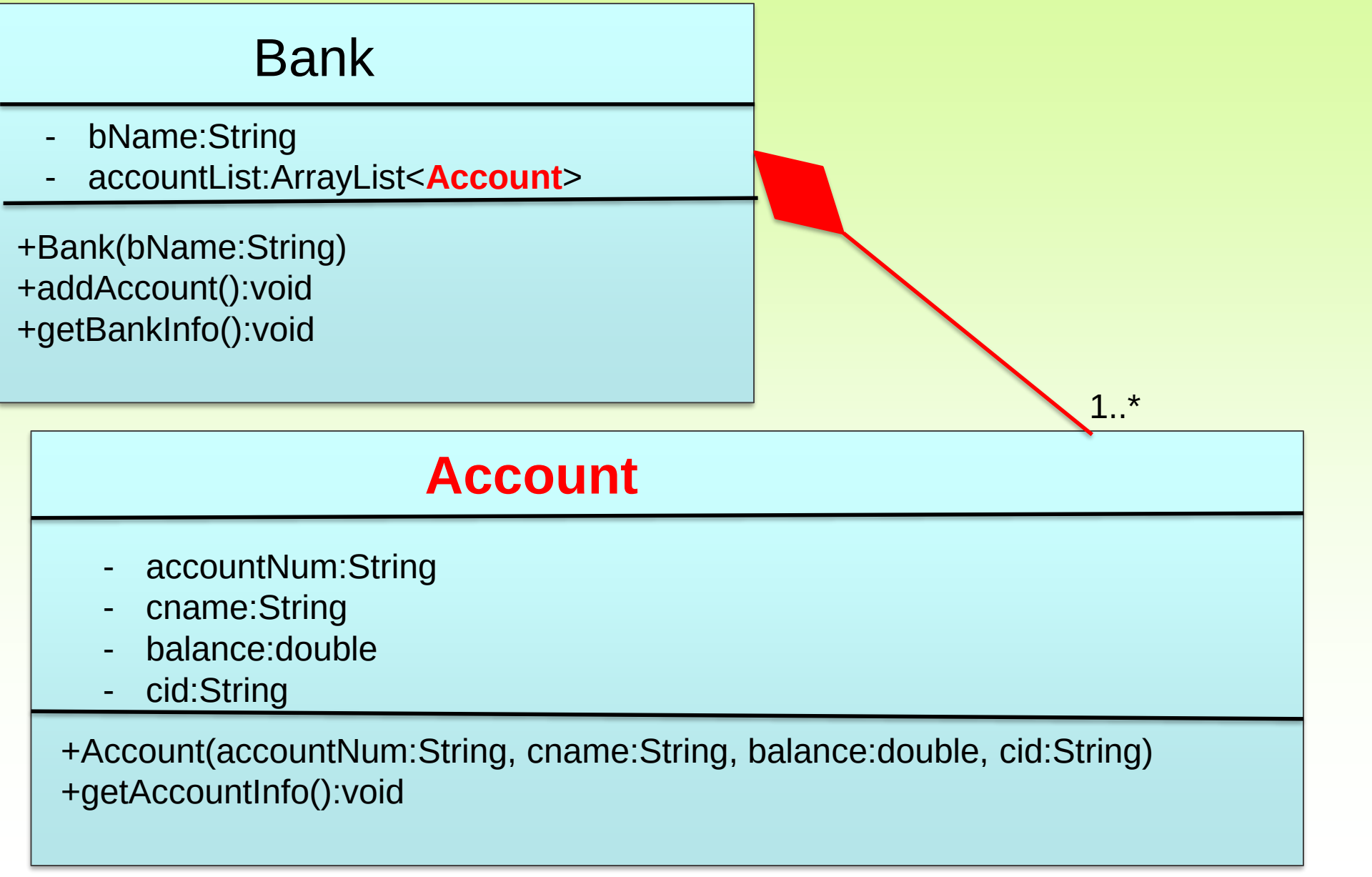


لتوضيح علاقة Composition في الجافا

اكتب برنامجا بلغة الجافا للتوصيف التالي:

- **تعريف صف Account** يحتوي:
  - الحقول التالية: رقم الحساب accountNum، اسم مالك الحساب cname، قيمة الرصيد balance، و الرقم الوطني لمالك الحساب cid
  - باني مناسب للتوصيف
  - طريقة getAccountInfo() والتي تطبع معلومات الحساب
- **تعريف صف Bank و هو مشكل من مجموعة حسابات** يحتوي:
  - ❖ الحقل اسم البنك bName والحقل AccountList والذي هو عبارة عن قائمة من الحسابات
  - ❖ باني مناسب للتوصيف
  - ❖ الطرق التالية:
  - ❖ addAccount() إضافة حساب للبنك حيث يتم ادخال معلومات الحساب من خلال لوحة المفاتيح
  - ❖ getAllAccounts() عرض معلومات كل الحسابات في البنك
- **تعريف الصف BankAppComposition** والذي يتم فيه نشاء غرض من الصف Bank واستدعاء الطرق الموجودة فيه

تمثيل علاقة الـ Composition باستخدام لغة UML



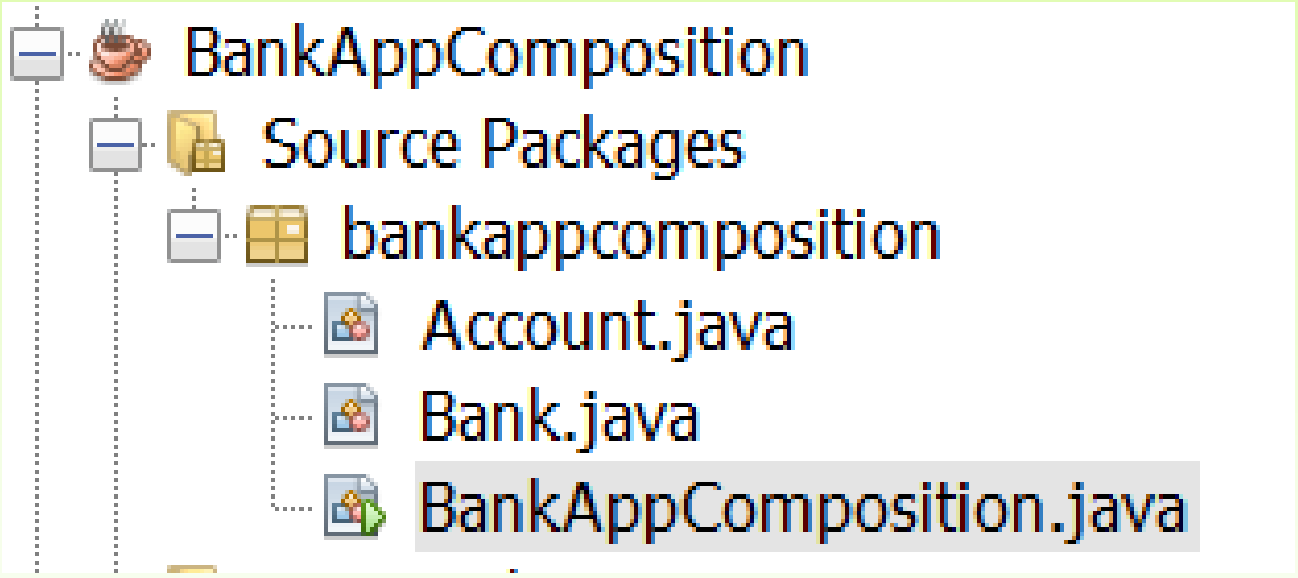
لتنفيذ علاقة Composition في الجافا، فإنه في الصف Bank

- أولا:

يتم التصريح عن الحقل accountList في الصف Bank على أنه قائمة متجهية ArrayList كل عنصر فيها من نوع الصف Account، و هذا الحقل حاليا خالي لا يحمل أي عنوان (أي لا يؤشر على أي غرض)

- ثانيا:

ضمن باني الصف Bank يتم إنشاء قائمة متجهية من النوع accountList تحوي أغراض من نوع Account، و جعل الحقل accountList يؤشر على (يحمل عنوان) القائمة المتجهية


BankAppComposition
Source Packages
bankappcomposition
Account.java
Bank.java
BankAppComposition.java

الصف Account

```
package bankappcomposition;
public class Account {
    private String accountNum;
    private String cname;
    private double balance;
    private String cid;
    public Account(String accountNum,String cname,double balance,String cid) {
        this.accountNum=accountNum;
        this.cname=cname;
        this.balance=balance;
        this.cid=cid;
    }
    public void AccountInfo(){
        System.out.println("Account Number= "+accountNum+
                           " Customer Name= "+cname+
                           " Balance="+balance+" CID="+cid);
    }
}
```

الصف Bank

```java
package bankappcomposition;
import java.util.*;
public class Bank {
    private String bName;
    private ArrayList<Account> accountList;
    public Bank(String bName){
        this.bName=bName;
        accountList=new ArrayList<Account>();
    }
    public void addAccount(){
        System.out.println("**********************");
        Scanner s=new Scanner(System.in);
        System.out.print("Enter Account Number=");
        String accountNum=s.nextLine();
        System.out.print("\nEnter Customer Name=");
        String cname=s.nextLine();
         System.out.print("\nEnter Balance=");
        double balance=s.nextDouble();
         System.out.print("\nEnter Customer ID=");
        String cid=s.next();
        Account ac=new Account(accountNum, cname, balance,cid);
        accountList.add(ac);
    }
     public void getBankInfo(){
        System.out.println("Information Regarding "+bName+" Bank");
        for(int i=0; i<accountList.size();i++)
            accountList.get(i).AccountInfo();
    }
}
```

لاحظ أنه عندما يتم انشاء غرض من الصف Bank سيتم انشاء ArrayList من أغراض من الصف Account داخله

لاحظ أن وجود الحساب متعلق بوجود البنك

الصف BankAppComposition

```
package bankappcomposition;
public class BankAppComposition {
    public static void main(String[] args) {
        Bank bank1=new Bank("Audi");
        bank1.addAccount();
        bank1.addAccount();
        bank1.getBankInfo();
    }
}
```

التنفيذ

****
Enter Account Number=111

Enter Customer Name=Rami Ali

Enter Balance=1000000

Enter Customer ID=123
****
Enter Account Number=222

Enter Customer Name=Sami Hussam

Enter Balance=200000

Enter Customer ID=456
Information Regarding Audi Bank
Account Number= 111 Customer Name= Rami Ali Balance=1000000.0 CID=123
Account Number= 222 Customer Name= Sami Hussam Balance=200000.0 CID=456

ملاحظات حول الصف Bank

- السطر التالي

```
private ArrayList<Account> accountList;
```

يعني التصريح عن الحقل accountList في الصف Bank على أنه ArrayList كل عنصر فيها من نوع الصف Account، و هذا الحقل حاليا خالي لا يحمل أي عنوان (أي لا يؤشر على أي غرض)

- السطر accountList=new ArrayList<Account>(); في الباني للصف Bank

```
public Bank(String bName){
    this.bName=bName;
    accountList=new ArrayList<Account>();
}
```

يعني أنه ضمن باني الصف Bank يتم إنشاء قائمة متجهية ArrayList تحوي أغراض من نوع Account، و جعل الحقل accountList يؤشر على (يحمل عنوان) القائمة

ArrayLists
القوائم المتجهية

Dr. REEMA AL-KAMHA

The `ArrayList` Class

- القائمة المتجهية AraayList هو صف يمكن إنشاء أغراض منه تستخدم لتخزين لائحة من البيانات التي كل منها من نوع Object (النوع Object هو النوع الأعم في الجافا)
- تختلف القائمة المتجهية ArrayList عن المتجهة Array من حيث:
  - يجب أن تكون عناصر المتجهة Array من نوع واحد، و لا يسمح بمزج عدة أنواع في متجهة واحدة
  - المتجهة بنية ثابتة لا يمكن تغيير الطول المحدد لها (زيادة أو نقصان) أثناء تنفيذ البرنامج. بينما القائمة المتجهية ArrayList لها بنية ديناميكية يمكن زيادة و نقصان طولها أثناء تنفيذ البرنامج

استخدام الصف Arraylist

- القائمة المتجهية ArrayList هي صف من الحزمة java.util ، لذلك للتعامل مع القائمة المتجهية، نكتب:

```
import java.util.ArrayList;
```

أو:

```
import java.util.*;
```

- يمكن تعريف قائمة متجهية alist من خلال استخدام الصف ArrayList كما يلي:

ArrayList alist=new ArrayList();

حيث تنشئ هذه التعليمة غرض alist ، و هذا الغرض سيخزن أغراض وليس أنواع بسيطة primitive Types. قد تكون الأغراض المضافة من أنواع مختلفة.

- لتعريف قائمة متجهية alist جميع عناصرها أغراض من نفس الصف، على سبيل المثال من نوع الصف String نكتب

ArrayList <String> alist=new ArrayList<String> ();

- لتعريف قائمة متجهية alist جميع عناصرها أغراض من الصف Book نكتب:

```
ArrayList<Book> alist = new ArrayList<Book>();
```

الصف Arraylist

- يبدأ ترقيم العناصر في القائمة المتجهية بدءا من الدليل رقم 0

Methods in the `ArrayList` Class

`alist.add( Object element)`

تضيف العنصر element إلى نهاية alist

`alist.add(int index, Object element)`

تضيف العنصر element إلى alist في اعتبارا من الموقع index

`alist.remove(int index)`

تحذف من alist العنصر الموجود في الموقع index

`alist.remove(object element)`

تحذف أول ظهور للعنصر element في حال وجوده تعيد true عندما يتحقق ذلك

Methods in the `ArrayList` Class

```
alist.clear()
```

تحذف كل العناصر من alist

```
alist.get(int index)
```

تعطي العنصر في الموقع index

```
alist.size()
```

تعيد عدد العناصر في alist

Methods in the `ArrayList` Class

```
alist.indexOf(object element)
```

تعيد دليل العنصر element في alist، و 1- لم يوجد العنصر

```
alist.contains(object element)
```

تعيد true إذا كانت alist تحوي العنصر element

```
Alist.isEmpty()
```

تعيد true إذا كانت alist فارغة

ملاحظة

- In Java, generic specifications can be used only with object types and not with primitive types. Thus, while it is perfectly legal to write a definition like

```
ArrayList<String> names = new ArrayList<String>();
```

it is not legal to write

```
ArrayList<int> numbers = new ArrayList<int>();
```

- To get around this problem, Java defines a ***wrapper class*** for each of the primitive types:

`boolean`	`Boolean`	`float`	`Float`
`byte`	`Byte`	`int`	`Integer`
`char`	`Character`	`long`	`Long`
`double`	`Double`	`short`	`Short`

```
import java.util.ArrayList;        // ArrayList هنا قمنا باستدعاء الكلاس
 public class Main {
    public static void main(String[] args) {
        // al  اسمه ArrayList هنا قمنا بإنشاء الغرض من الصف
       ArrayList al = new ArrayList();
        //       هنا قمنا بعرض عناصر الغرض al و عرض عددهم
       System.out.println("All elements:       " + al);
       System.out.println("Number of elements: " + al.size() + "\n");
        // al هنا قمنا بإضافة 5 عناصر في الغرض
       al.add("A");   al.add("B");   al.add("C");  al.add("D");  al.add("E");
        //  هنا قمنا بعرض عناصر الكائن al و عرض عددهم
       System.out.println("All elements:       " + al);
       System.out.println("Number of elements: " + al.size() + "\n");
        //  1 رقمindex  العنصر الموجود على الـ B  هنا قمنا بحذف العنصر الذي يملك القيمة
       al.remove("B");      al.remove(1);
        //    هنا قمنا بعرض عناصر الكائن alو عرض عددهم
       System.out.println("All elements:       " + al);
       System.out.println("Number of elements: " + al.size() + "\n");
    }
 }
```

البرامج المطلوب قراءتها و تنفيذها و فهمها

- StudentAggregation
- DepartmentStudentAggregation
- BankAppComposition
- LibrarySystem
- ArrayListApplication