



Interfaces in Java

الواجهات في الجافا

Dr. REEMA AL-KAMHA

مقدمة

- تعتبر الواجهة و كأنها صف مجرد بالكامل (Full Abstract) حيث:
 - كل الطرق فيها عامة و مجردة **public abstract** بشكل تلقائي، حتى و إن لم يذكر ذلك بشكل صريح.
 - كل المتغيرات في الواجهة تعتبر بشكل تلقائي **public static final** حتى و إن لم يذكر ذلك بشكل صريح.
- تعرف الواجهة طرناً تتألف فقط من ترويسة الطريقة بدون ذكر أي تفاصيل للكود البرمجي للطريقة. يتم تعريف جسم الطريقة في الصفوف التي تقوم بتبني (بتنفيذ) (implements) هذه الواجهة.
- يقوم الصف الذي يتبنى (ينفذ) (implements) الواجهة بإعادة كتابة الطريقة (override) و ذلك بتعريف جسم الطريقة و تفاصيل الكود البرمجي لها.
- يمكن للصف أن يتبنى (ينفذ) (implements) عدة واجهات، و لكن يمكنه أن يكون يوسع (extends) صف واحد فقط.

مقدمة

- تعتبر الواجهة بمثابة مخطط عام للصف الذي يتبناها (ينفذها) (implements) من خلال التصريح عن الطرق التي ترغب بتوفيرها في الصف. أي تحدد الواجهة ما يجب أن يفعله الصف، ولكن لا تحدد كيفية الفعل.
- تجعل الواجهة العلاقة بين الصفوف غير مباشرة: عن طريق الواجهة فقط مما يجعل الصفوف أكثر استقلالية عن بعضها البعض.

إنشاء واجهة interface

تستخدم الكلمة المفتاحية **interface** لإنشاء واجهة بلغة java كما يلي :

```
interface Name {
```

المقصود بها اسم
الواجهة الذي
نريد

يتم هنا التصريح عن المتغيرات و مجموعة الطرق التي
تعرفها الواجهة

```
}
```

مثال

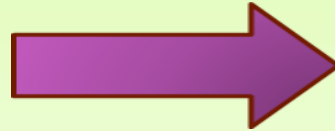
يعرف المقطع البرمجي التالي واجهة اسمها A توفر طريقة تسمى getInfo

```
interface A{
```



تم التصريح عن
واجهة تسمى A

```
void printA();
```



الطريقة printA التي تعرفها الواجهة A
وهي public abstract بشكل تلقائي
على الرغم من أننا لم نشر بشكل صريح

```
}
```

نلاحظ انه تم
التصريح عن
ترويسة الدالة
فقط ولم يتم
تعريف جسمها

الأشياء التي يمكن تعريفها داخل الواجهة

- طرقاً لا تملك جسم أي طرق مجردة، حيث أن أي طريقة يتم تعريفها بداخل الواجهة تعتبر `public abstract` حتى لو لم تعرفها كـ `public`، ولا يمكن تعريف الطرق في الواجهة كـ `private` أو `protected` أو `final` أو `static`. السبب في ذلك:

○ وجود صفوف تقوم بتنفيذ هذه الواجهة، و تقوم بإعادة كتابة كل الطرق الموجودة في الواجهة بما يناسب الصف.

- يمكن للواجهة أن تحتوي على متغيرات، ولكن أي متغير يتم تعريفه سيكون بشكل تلقائي متغير صف (أي `static`) و ثابت (أي `final`)، أي سيكون `public static final` حتى ولو لم يذكر ذلك صراحة، الأمر الذي يجعلك مجبراً على إعطاءها قيمة مباشرة عند تعريفها مع عدم إمكانية تغيير هذه القيمة.

○ ملاحظة: يجب وضع قيمة للثوابت و إلا يظهر المترجم رسالة خطأ.

○ ملاحظة: الحقل (المتغير) في الصف الذي من نوع `static final` له نسخة واحدة في الذاكرة لا يمكن تغييرها.

- A field that is both static and final has only one piece of storage that cannot be changed.

مثال

يبين المثال التالي واجهة تحوي عدد من الثوابت، ونلاحظ أنه يجب وضع قيم هذه الثوابت وإلا سيظهر المترجم خطأ

```
interface A{  
    public static final double x=4.3;  
    int y=2;  
}
```

يعتبر المتغير y كـ `public static final` حتى وإن لم نذكر ذلك بشكل صريح

لا نستطيع تغيير قيمة
المتغيرين
 x و y

كيفية ترجمة طرق و متغيرات الواجهة

```
interface Printable{  
int MIN=5;  
void print();  
}
```

Printable.java

compiler

```
interface Printable{  
public static final int MIN=5;  
public abstract void print();  
}
```

Printable.class

ملاحظة

- لا يمكن تعريف الواجهة كـ **private** أو **protected** لأنه دائماً تعتبر **public** حتى وإن لم نضع كلمة **public** قبلها.
- كما أنه لا يمكن تعريف الواجهة كـ **final** أو **static** لأنه تم تصميم الواجهة لجعل أي صف يتمكن من تنفيذها يقوم بإعادة تعريف (override) الطرق الموجودة فيها.

كيفية التعامل مع الواجهة

❖ حتى تكون الواجهة مفيدة ولكي يتم التعامل معها يجب أن يتبناها (ينفذها) صف معين وذلك بكتابة التعليمات البرمجية لكل طريقة في الواجهة ضمن الصف الذي يقوم بتبنيها (بتنفيذها) و إلا سيعطي المترجم خطأ.

□ يمكن للمبرمج أن يعرف جسماً دون تعليمات للطريقة التي لا يريد كتابة تعليمات برمجية لها.

❖ لجعل صف ما ينفذ واجهة، تستخدم الكلمة المفتاحية `implements` والتي تشير إلى أن صفاً ما تبني الواجهة.

مثال عن واجهة و صف يتبنى (ينفذ) هذه الواجهة

```
public interface A {  
void printA();  
}
```

واجهة A والتي تصرح عن الطريقة printA

للتعامل مع الواجهة A ، نكتب الصف B الذي يتبنى (ينفذ) هذه الواجهة من خلال الكلمة المفتاحية implements ، و يقوم بتعريف التعليمات التنفيذية للطريقة printA

```
Public class B implements A{  
@Override  
void printA(){  
System.out.print("Interface");  
}
```

عرفنا صف B يتبنى (ينفذ) الواجهة A حيث يجب أن يعيد هذا الصف كتابة كل الطرق الموجودة في الواجهة

ملاحظة

❖ عندما يتبنى (ينفذ) صف معين واجهة، فإن هذه الواجهة تصبح نوعا يمكن أن يُوَشر إلى هذا الصف. هذه الميزة مهمة للغاية:

➤ لأنها تسمح بتعريف متغير مرجعي من نوع الواجهة يمكن أن يُوَشر إلى أي صف يتبنى (ينفذ) الواجهة، و بالتالي يمكن استدعاء أي طريقة مصرح عنها في الواجهة (و التي تم إعادة كتابتها في الصف الذي يتبنى الواجهة) باستخدام المتغير المرجعي نفسه.

توسيع الواجهات بالوراثة

Extending Interfaces with Inheritance

- يمكن توسيع واجهة معينة باستخدام مفهوم الوراثة، حيث يمكن لواجهة معينة أن ترث من واجهة أخرى أو أكثر، وهذا ما يعرف بمصطلح **بالوراثة المتعددة Multiple Inheritance والتي** **تعني أنه يمكن أن يكون للواجهة أكثر من أب مباشر واحد.**
أي أن:

الوراثة المتعددة مسموحة في حالة الواجهات في لغة java .

تذكر

الوراثة المتعددة في جافا
(أي صف له أكثر من صف
أب مباشر) غير ممكنة
بالنسبة للصفوف أي لا تسمح
لغة جافا للصف الواحد أن
يرث من أكثر من صف
مباشر

الوراثة المتعددة باستخدام الواجهات

Multiple Inheritance with Interfaces

- نعلم أن الوراثة المتعددة غير مسموحة بلغة الجافا لأنها تقود إلى بعض المشاكل.

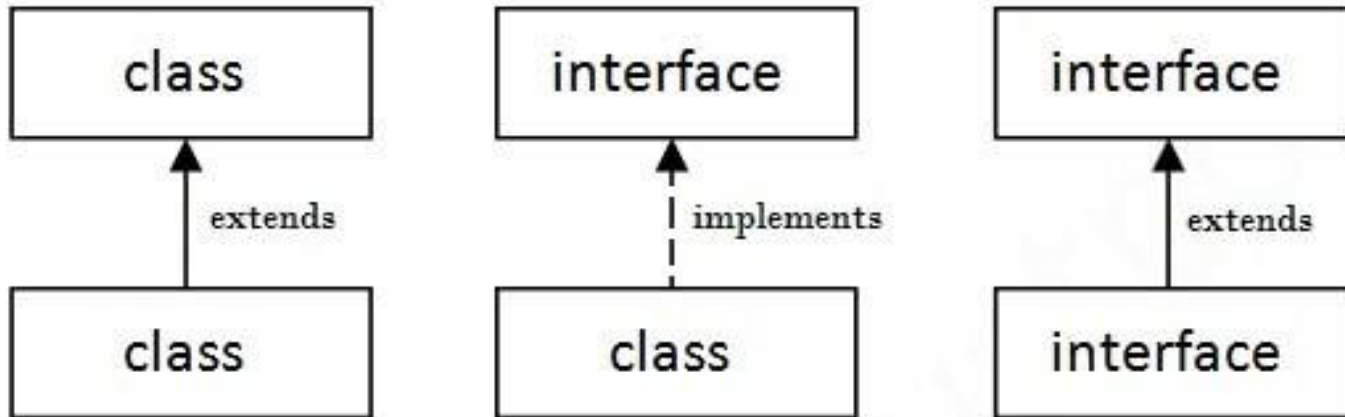
- أهم هذه المشاكل هي إمكانية وجود الطريقة نفسها في أكثر من صف أب. في هذه الحالة، لا يستطيع المترجم عند استدعاء هذه الطريقة المشتركة في صف الابن تحديد صف الأب الذي يجب أن تستدعي منه الطريقة (لأنها موجودة في أكثر من صف أب).

الوراثة المتعددة باستخدام الواجهات

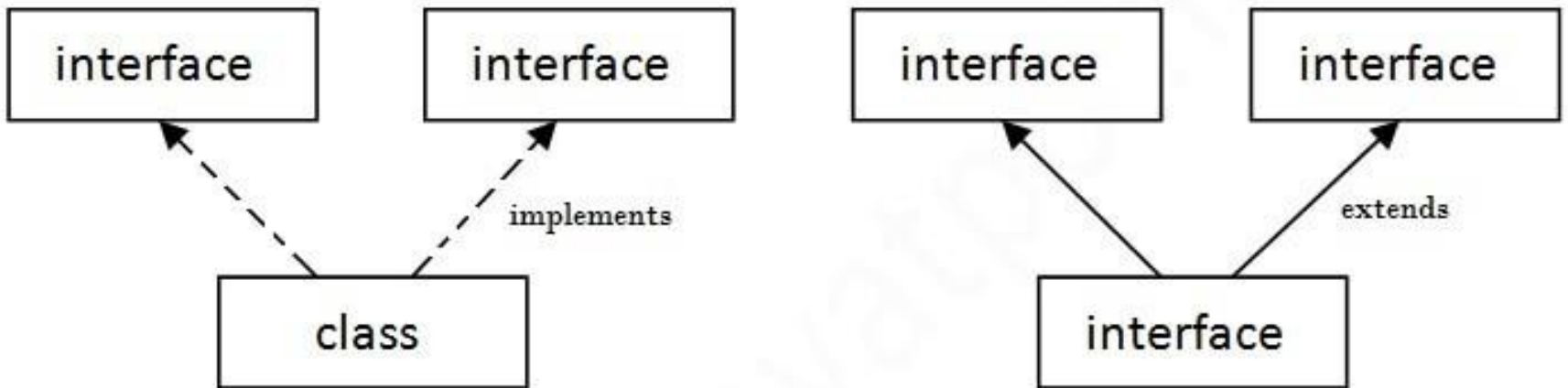
Multiple Inheritance with Interfaces

- على الرغم من أننا لا نستطيع تعريف وراثة متعددة بين الصفوف في لغة الجافا، إلا أننا نستطيع أن نعرف الوراثة المتعددة بشكل غير مباشر باستخدام مفهوم الواجهة. حيث يمكن للصف الواحد أن يرث من صف واحد فقط، و يتبنى (ينفذ) أي عدد من الواجهات.
- بما أننا نستطيع أن نشير إلى الصف من خلال الواجهات التي يتبناها هذا الصف، فإننا نستطيع أن نقول إن هذا الصف هو من نوع الواجهة الأولى و الواجهة الثانية و الواجهة الثالثة و هلم جرا. و هذا هو جوهر الوراثة المتعددة.

تمثيل الواجهات في لغة UML

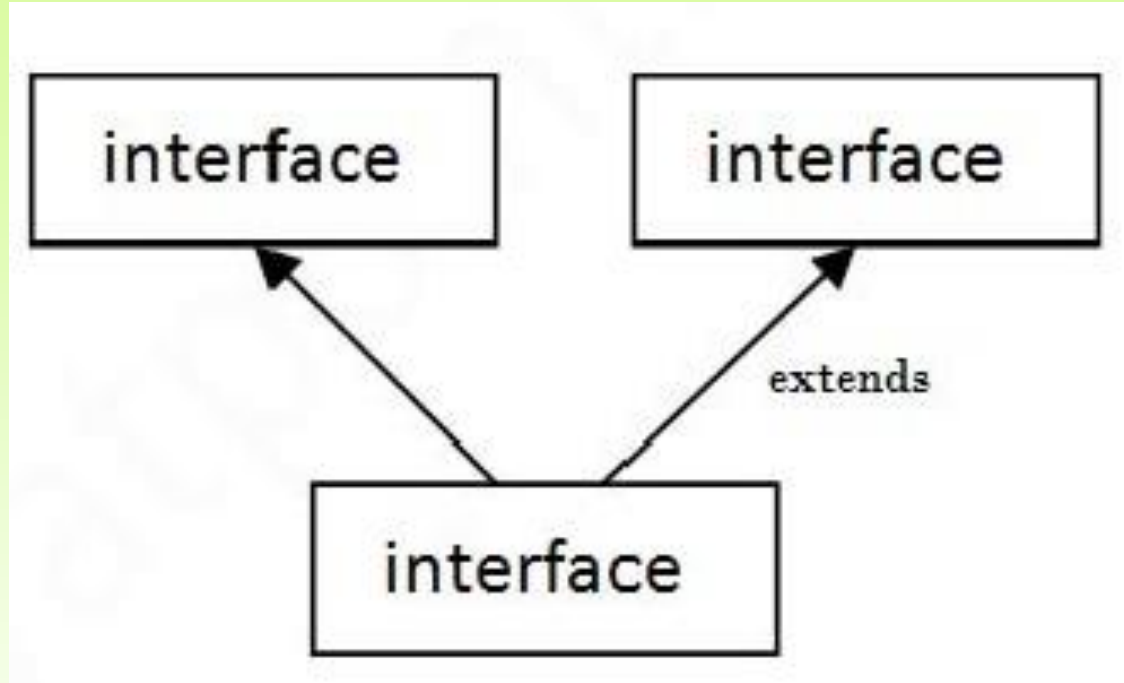


الوراثة المتعددة في الجافا



Multiple Inheritance in Java

يوضح الشكل التالي الوراثة المتعددة بين الواجهات



ملاحظة:

كلمة `extends` تستخدم للوراثة بين الواجهات

مثال عن واجهة ترث من واجهتين

interface A { }



عرفنا واجهة A

interface B { }



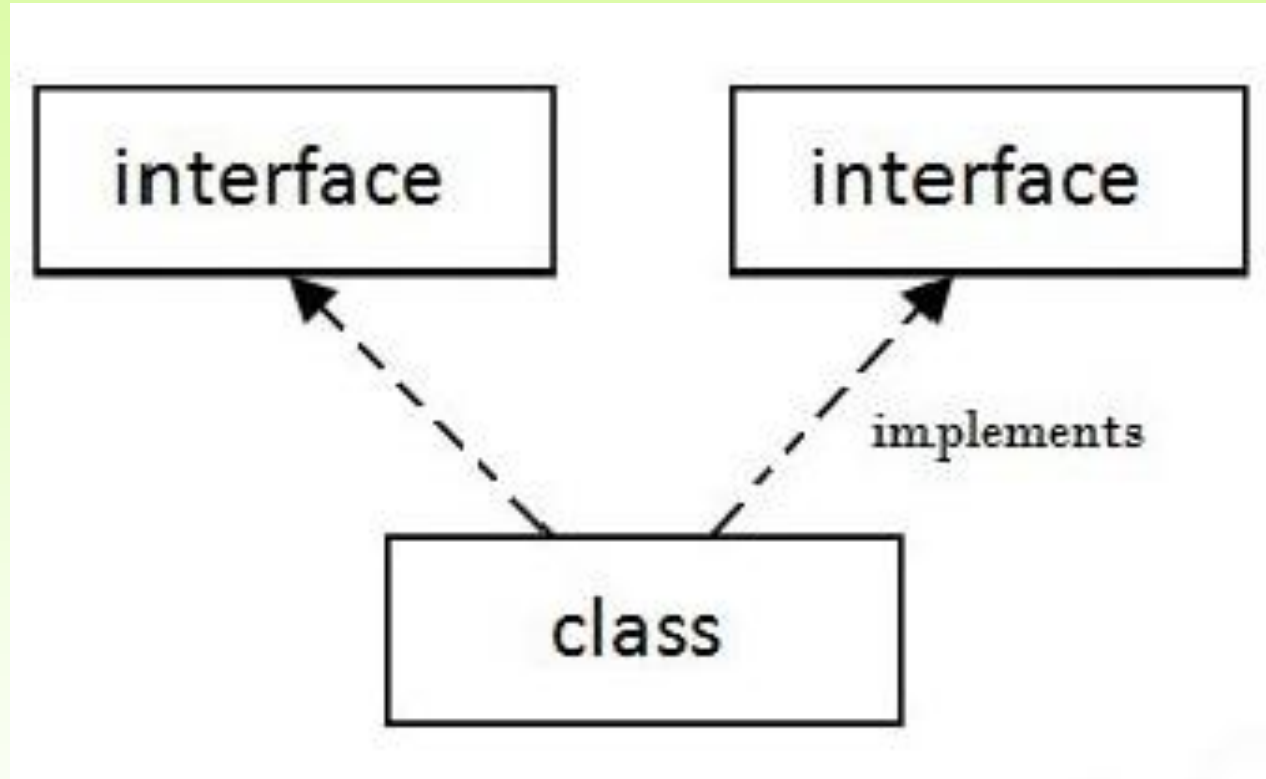
عرفنا واجهة B

**interface C extends A, B {
}**



عرفنا واجهة C ترث من
الواجهتين A و B معا

صف یتبئی (ینفذ) واجهتین



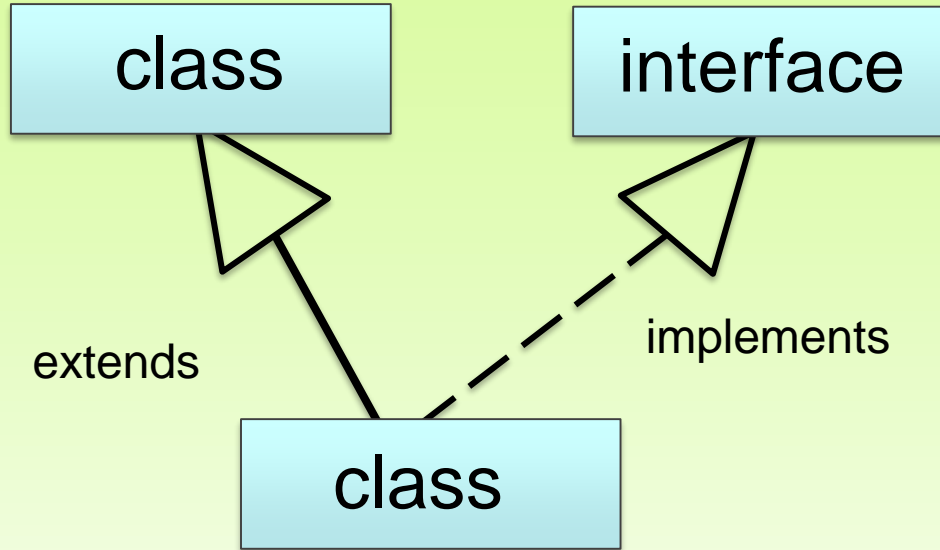
مثال عن صف يتبنى (ينفذ) واجهتين

interface A { } → واجهة A

interface B { } → واجهة B

class C implements A, B { } → واجهة C تتبنى تنفيذ كلا من الواجهتين A و B

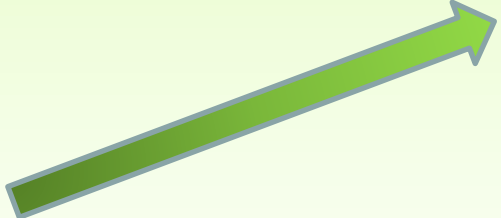
صف يتبنى (ينفذ) واجهة و يرث من صف آخر



مثال عن صف يتبنى (ينفذ) واجهة و يرث من صف آخر

interface A { }  واجهة A

class B { }  واجهة B

 الصف C، يرث الصف B ويتبنى تنفيذ الواجهة A

class C extends B implements A { }

ملاحظات عامة عند كتابة الواجهة

- لا يستخدم أي محدد وصول **Access Modifer** عند تعريف الواجهة.
- لا يستخدم أي محدد وصول **Access Modifer** عند تعريف طريقة داخل الواجهة.
- الطرق داخل الواجهة لا تحوي أي جسم، فقط ترويسة الطريقة.
- لا تملك الواجهة باني (constructor).

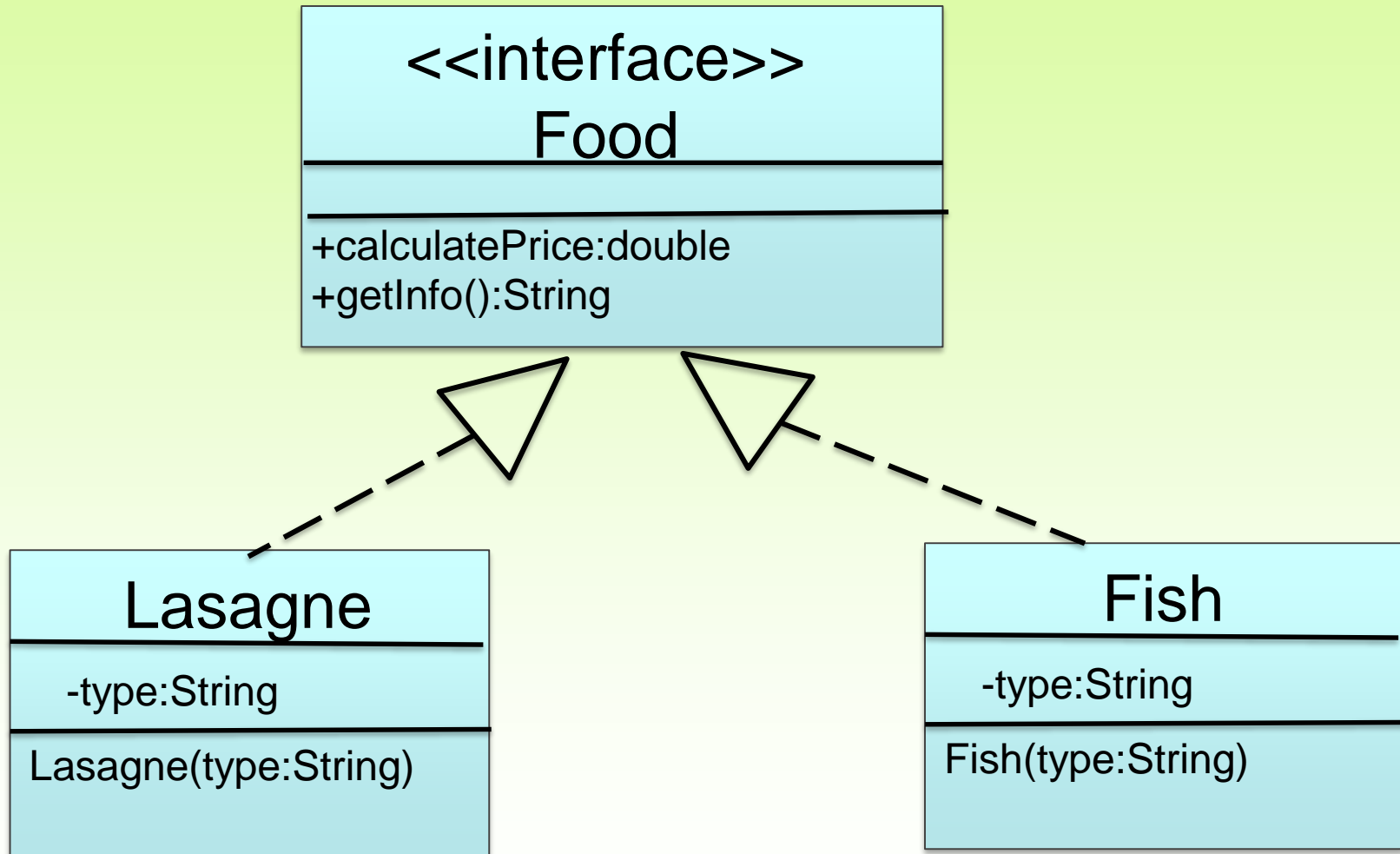
اكتب برنامج بلغة جافا تتعامل من خلاله مع واجهة Food يصرح من خلالها عن طريقة لحساب السعر calculatePrice و طريقة getInfo لإعادة المعلومات.

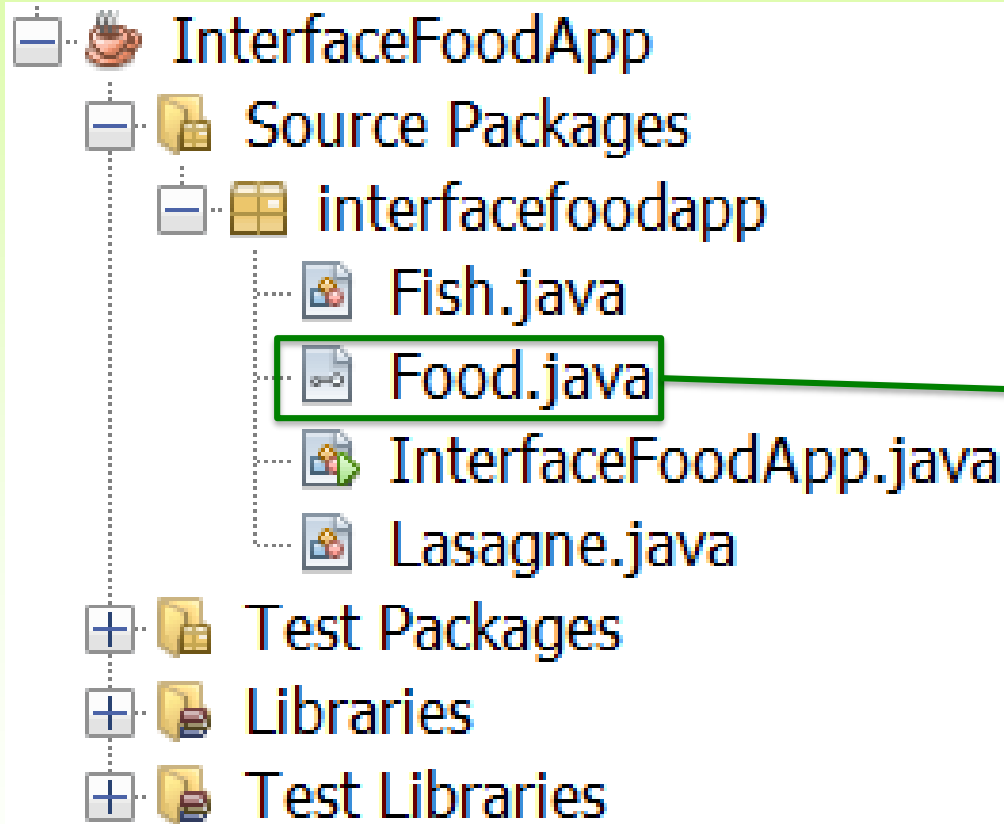
○ يتبنى الواجهة صفان Fish و Lasagne.

○ يحوي الصف Fish الحقل type، إضافة لباني مناسب للتوصيف. إذا كان السمك مقليا فثمنه 2000 و إذا كان مشويا فثمنه 3000.

○ يحوي الصف Lasagne الحقل type، إضافة لباني مناسب للتوصيف. إذا كانت اللازانيا بالخضار فثمنها 1500 و إذا كانت بدون خضار فثمنها 2000.

تمثيل الواجهة باستخدام UML





واجهة

الواجهة Food

تكون الواجهة public حتى وإن لم نعرفها كذلك

```
package interfacefoodapp;  
interface Food {  
    double calcuatePrice ();  
    String getInfo ();  
}
```

❑ لاحظ أن الطرق ضمن الواجهة ليس لها جسم، ترويسة فقط، وهي بشكل

تلقائي public

❑ سيتم إعادة كتابة كل الطرق الموجودة في الواجهة Food ضمن الصفوف

التي تتبناها (تنفذها)

الصف Fish الذي يتبنى (ينفذ) الواجهة Food

```
package interfacefoodapp;
public class Fish implements Food{
    private String type;
    Fish(String type){
        this.type=type;
    }
    @Override
    public double calcuatePrice() {
        double price=0;
        if(type.equals("Fried"))
            price=2000;
        else if(type.equals("Roasted"))
            price=3000;
        return price;
    }
    @Override
    public String getInfo() {
        return "The price of the " + type + " fish = " + calcuatePrice();
    }
}
```

الصف Fish ينبنى (ينفذ) الواجهة Food

إعادة كتابة الطريقة calculatePrice الموجودة في الواجهة Food

إعادة كتابة الطريقة getInfo الموجودة في الواجهة Food

الصف Lasagne الذي يتبنى (ينفذ) الواجهة Food

```
package interfacefoodapp;
```

```
public class Lasagne implements Food {
```

```
    private String type;
```

```
    Lasagne(String type) {
```

```
        this.type=type;
```

```
    }
```

```
    @Override
```

إعادة كتابة الطريقة calculatePrice الموجودة في الواجهة Food

```
    public double calculatePrice() {
```

```
        double price=0;
```

```
        if(type.equals("Veg"))
```

```
            price=1500;
```

```
        else if(type.equals("Nonveg"))
```

```
            price=2000;
```

```
        return price;
```

```
    }
```

```
    @Override
```

```
    public String getInfo() {
```

```
        return "The price of the " + type + " lasagn = " + calculatePrice();
```

```
    }
```

```
}
```

الصف Lasagne ينبنى (ينفذ) الواجهة Food

إعادة كتابة الطريقة calculatePrice الموجودة في الواجهة Food

إعادة كتابة الطريقة getInfo الموجودة في الواجهة Food

الصف الرئيسي InterfaceFoodApp

```
package interfacefoodapp;
public class InterfaceFoodApp {
    public static void main(String[] args) {
        Food food=new Fish("Roasted");
        System.out.println(food.getInfo());
        food=new Lasagne("Veg");
        System.out.println(food.getInfo());
    }
}
```

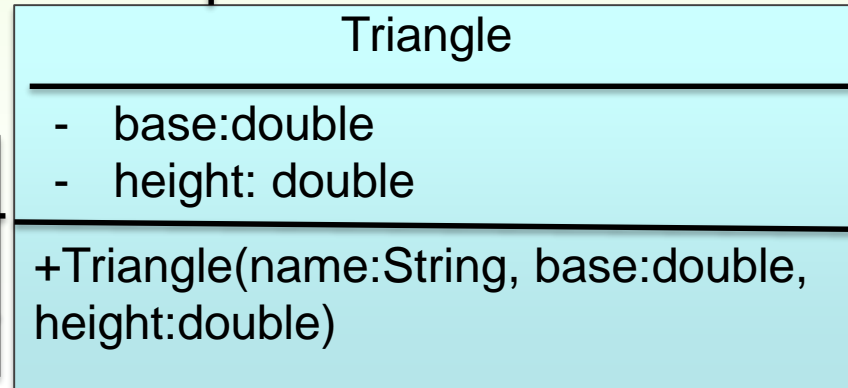
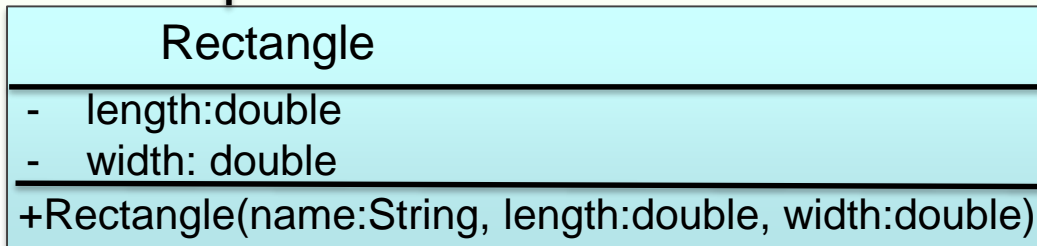
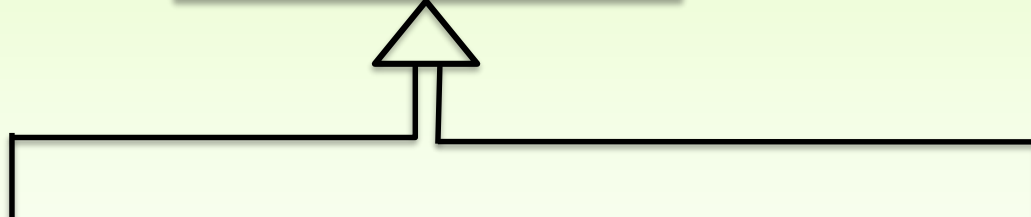
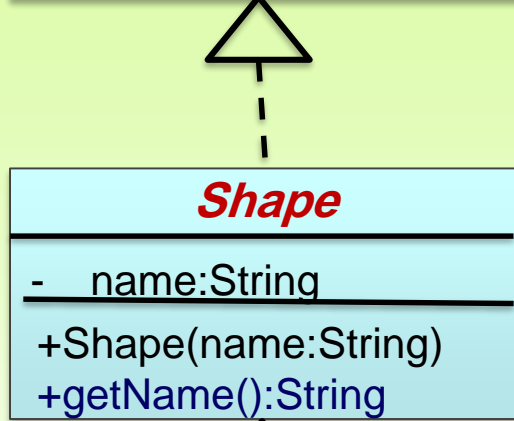
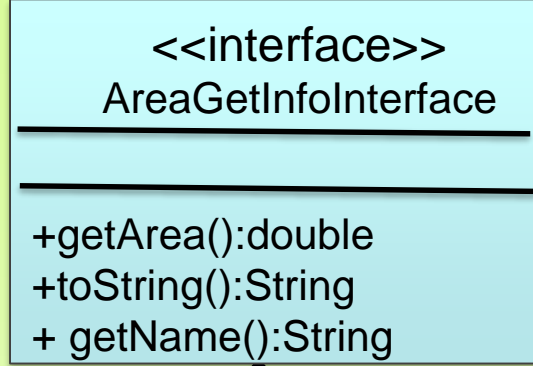
- أنشأنا متغير مرجعي food من نوع الواجهة Food، يُوْشر على الصفوف Fish و Lasagne التي (تتبناه) تنفذه
- لا يمكن انشاء أغراض من الواجهة Food

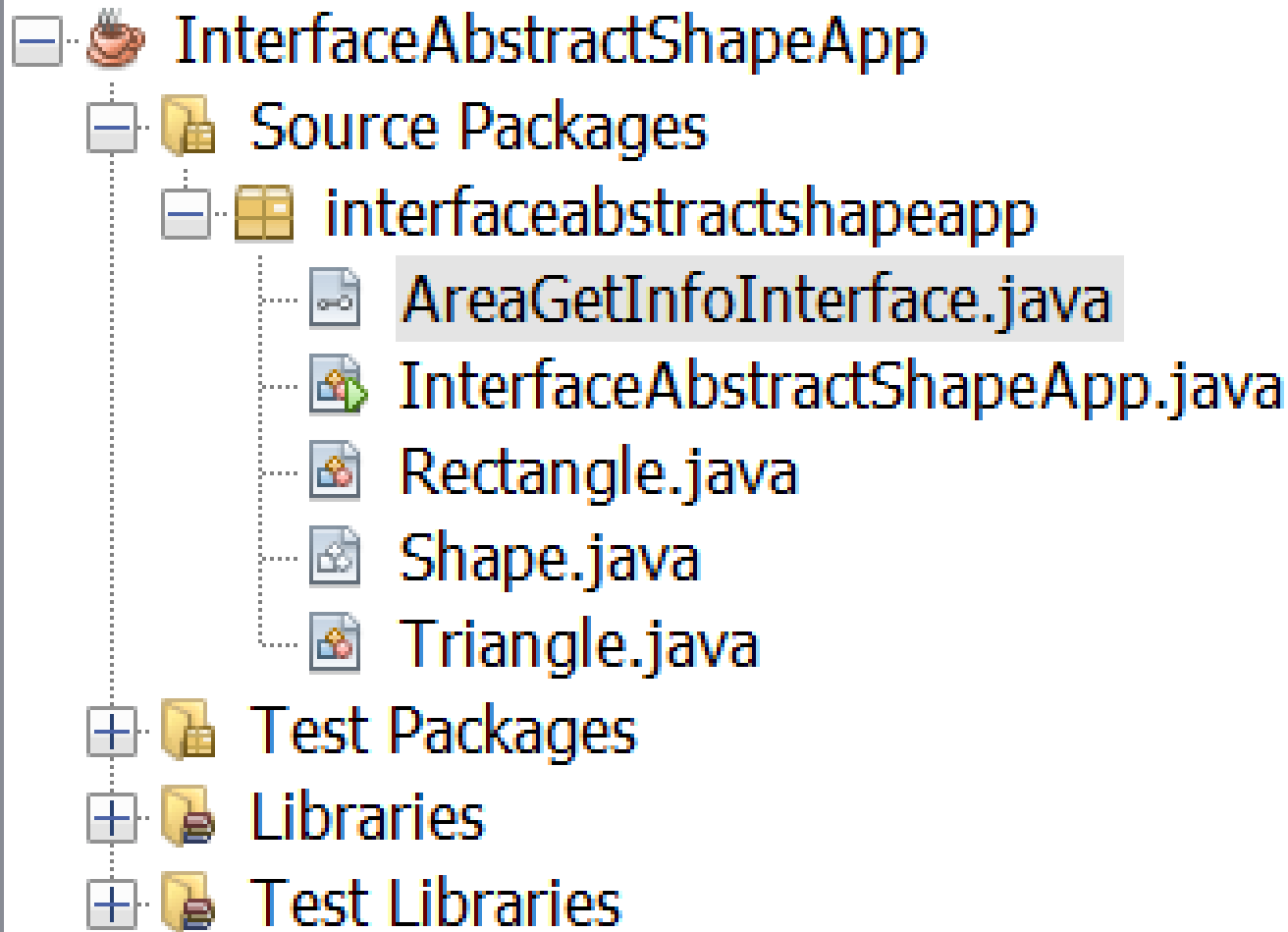
نتيجة التنفيذ

The price of the Roasted fish = 3000.0

The price of the Veg lasagn = 1500.0

InterfaceAbstractShapeApp





الواجهة AreaGetInfoInterface

```
package interfaceabstractshapeapp;  
interface AreaGetInfoInterface {  
    double getArea ();  
    String getInfo ();  
    String getName ();  
}
```

الصف المجرد Shape الذي يتبنى (ينفذ) الواجهة AreaGetInfoInterface

```
package interfaceabstractshapeapp;

public abstract class Shape implements AreaGetInfoInterface {
    private String name;
    public Shape(String name){
        this.name=name;
    }
    @Override
    public String getName(){
        return name;
    }
}
```

الصف Rectangle الذي يرث من الصف المجرد Shape

```
package interfaceabstractshapeapp;
public class Rectangle extends Shape {
    double length;
    double width;
    public Rectangle(String name, double length, double width) {
        super(name);
        this.length=length;
        this.width=width;
    }
    @Override
    public double getArea() {
        return length*width;
    }
    @Override
    public String getInfo() {
        return "The name is:"+ getName()+" Length="+length+
            " Width="+width+ " The area = "+getArea();
    }
}
```

الصف Triangle الذي يرث من الصف المجرد Shape

```
package interfaceabstractshapeapp;
public class Triangle extends Shape{
    private double base;
    private double height;
    public Triangle(String name, double base, double height){
        super(name);
        this.base=base;
        this.height=height;
    }
    @Override
    public double getArea(){
        return 0.5*base*height;
    }
    @Override
    public String getInfo(){
        return "The name is:"+getName()+ " Base=" +base+
            " Height="+ height+ " The area = "+getArea();
    }
}
```

InterfaceAbstractShapeApp

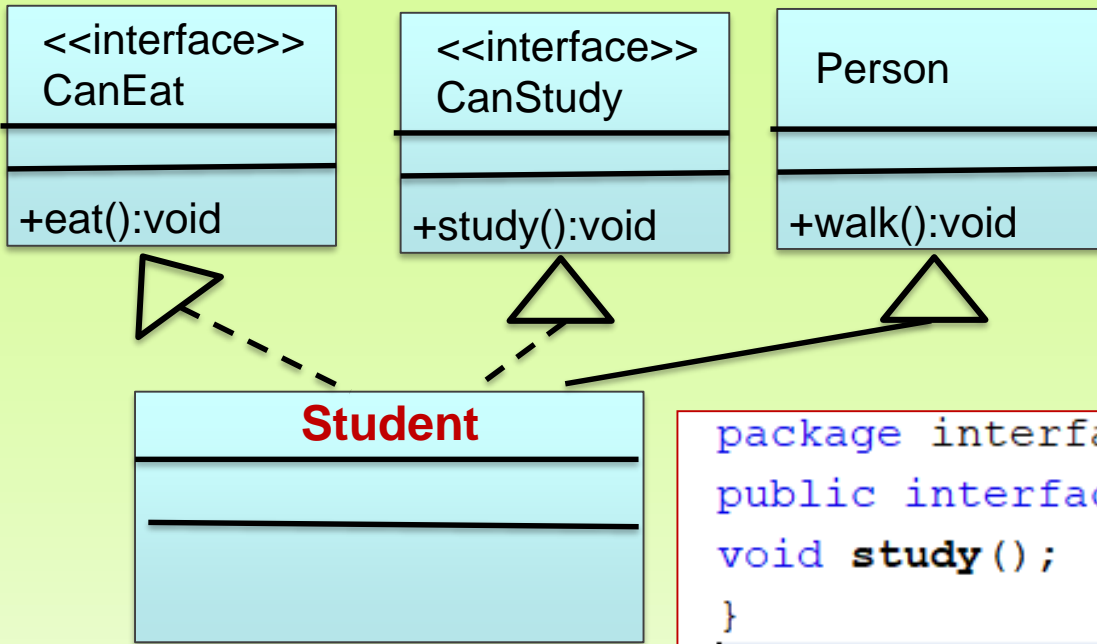
```
package interfaceabstractshapeapp;  
public class InterfaceAbstractShapeApp {  
    public static void main(String[] args) {  
        AreaGetInfoInterface r=new Rectangle("Rectangle",5,3);  
        System.out.println(r.getInfo());  
        r=new Triangle("Triangle",10,5);  
        System.out.println(r.getInfo());  
    }  
}
```


نتيجة التنفيذ

The name is:Rectangle Length=5.0 Width=3.0 The area = 15.0

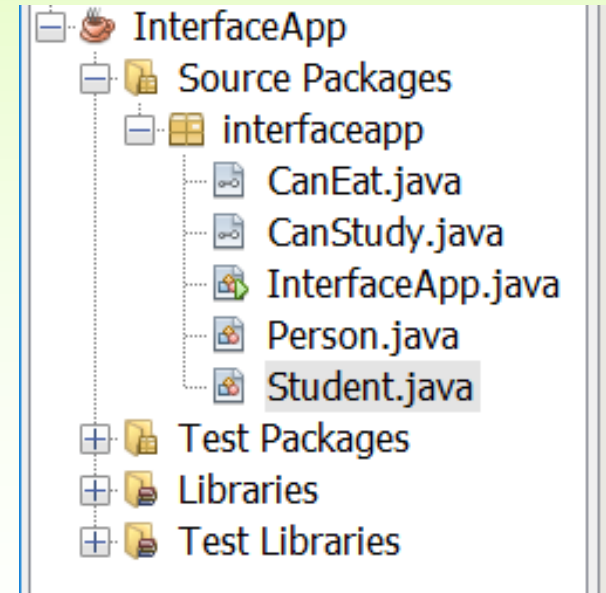
The name is:Triangle Base= 10.0 Height=5.0 The area = 25.0

تمثيل الواجهة لبرنامج InterfaceApp

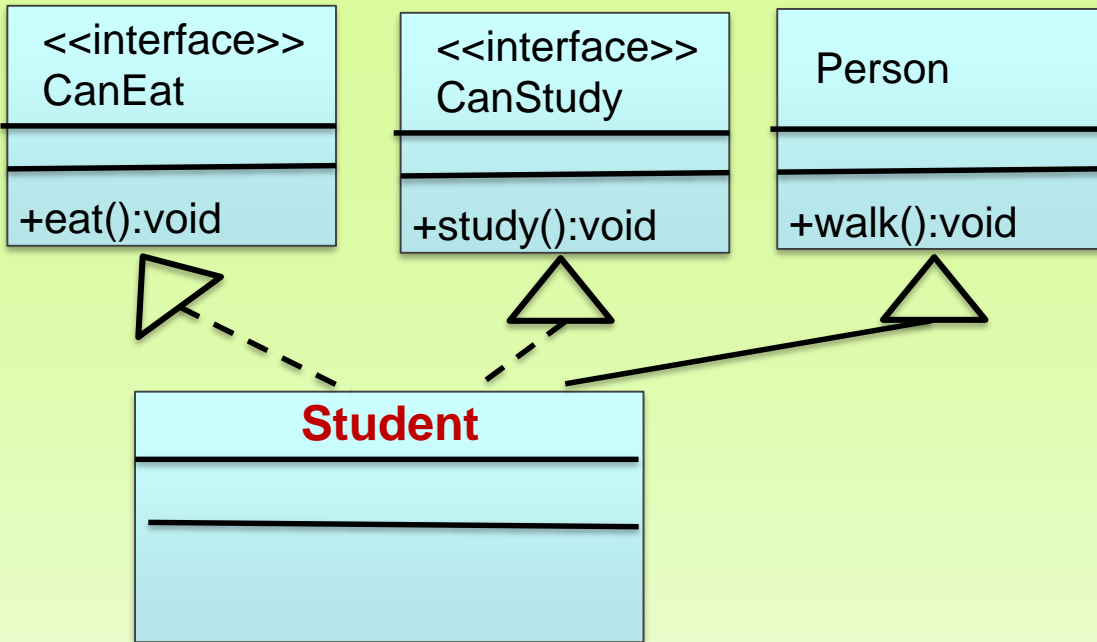


```
package interfaceapp;
public interface CanEat {
    void eat();
}
```

```
package interfaceapp;
public class Person {
    public void walk() {
        System.out.println("Person can walk");
    }
}
```

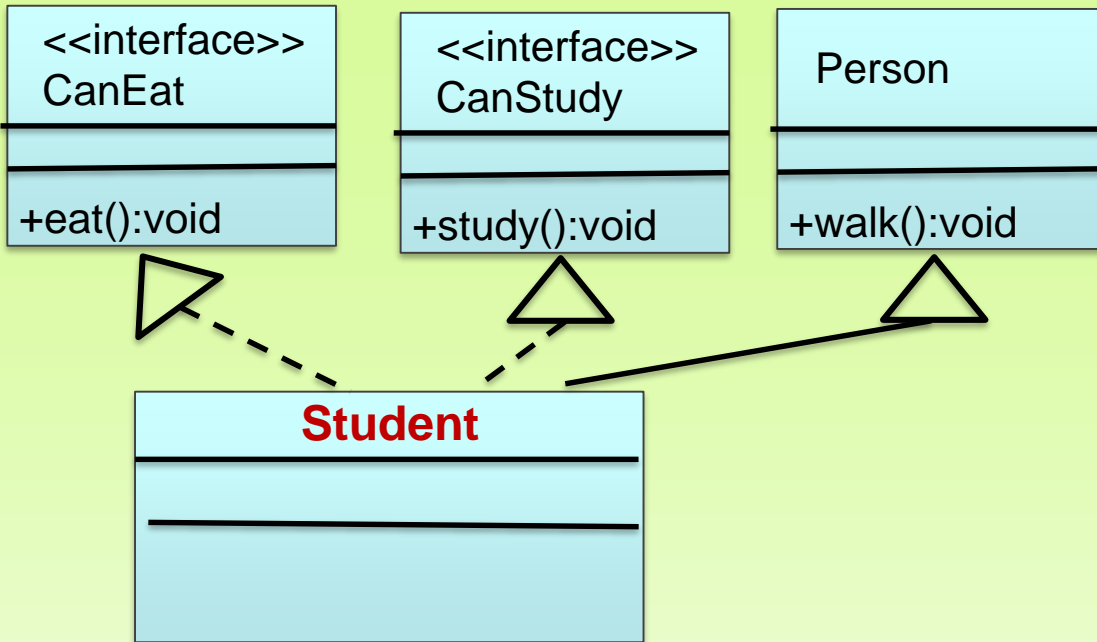


تمثيل الواجهة لبرنامج InterfaceApp



```
package interfaceapp;
class Student extends Person implements CanEat, CanStudy {
    @Override
    public void eat() {
        System.out.println("Student can eat");
    }
    @Override
    public void study() {
        System.out.println("Student can study");
    }
}
```

تمثيل الواجهة لبرنامج
InterfaceApp



```
package interfaceapp;
public class InterfaceApp {
    public static void main(String[] args) {
        Student s=new Student();
        s.eat();
        s.walk();
        s.study();
    }
}
```

نتيجة التنفيذ

Student can eat
Person can walk
Student can study

البرامج المطلوب قراءتها و تنفيذها و فهمها

- **InterfaceFoodApp**
- **InterfaceAbstractShapeApp**
- **InterfaceApp**