البرمجة الشيئية
Object oriented programming (with Java)

ITGS211

المحاضرة الحادية عشر

الفصل الدراسي خريف 2024/2023

Interface in Java
الإنترفيس في جافا

- **مفهوم الإنترفيس في جافا**
- **interface** هي كلمة محجوزة في جافا (تعني الربط او الوسيط).
- عمل مطوروا لغة جافا على ابتكار نوع جديد يشبه الكلاس العادي و يسمح لنا بتطبيق مبدأ تعدد الوراثة الذي يعتبر شيء مهم جداً في لغات البرمجة. هذا النوع الجديد يسمى **interface**.

الأشياء التي يمكن تعريفها بداخل الإنترفيس

- دوال مجردة (Abstract Method)، أي لا تملك body
- دوال عادية و لكن نوعها static
- متغيرات مع إعطائهم قيمة بشكل مباشرةً عند تعريفهم. لأن أي متغير تقوم بتعرفه بداخل الإنترفيس يعتبر معرف كـ public final static بشكل تلقائي.
- كلاسات متداخلة (Nested Classes)، أي كلاس نوعه static بداخل كلاس نوعه static
- إنترفيسات متداخلة (Nested Interfaces)، أي إنترفيس بداخل إنترفيس.

طريقة تعريف الإنترفيس

- الإنترفيس هو في الأساس **Full Abstract**، لكن لا يجب وضع الكلمة **abstract** عند تعريفه. و لا يمكن تعريف إنترفيس كـ **prviate** أو **protected** لأنه دائماً يعتبر **public** حتى لو لم تضع كلمة **public** قبله. كما أنه لا يمكن تعريف الإنترفيس كـ **final** أو **static** لأنه تم تصميم الإنترفيس لجعل أي كلاس يرثه يفعل **Override** للدوال الموجودة فيه.
- إذاً لتعريف إنترفيس، أكتب **interface** ثم ضع له أي اسم تريده.

```
interface MyInterface {}
```

طريقة تعريف متغيرات بداخل إنترفيس في جافا

- متغيرات الإنترفيس تعتبر معرفة كـ **public final static** حتى لو لم تقم بتعريفها كذلك، الأمر الذي يجعلك مجبراً على إعطاءها قيمة مباشرةً عند تعريفها مع عدم إمكانية تغيير هذه القيمة.

```
interface MyInterface {

   int X = 1;          // المتغير X يعتبر معرف كـ public final static حتى و لم نعرفه كذلك

}
```

طريقة تعريف دوال بداخل إنترفيس في جافا

- دوال الإنترفيس هي في الأساس **Abstract Method**، إذاً لا حاجة إلى وضع الكلمة **abstract** عند تعريف أي دالة بداخله. كما أن أي دالة يتم تعريفها بداخله تعتبر **public** حتى لو لم تعرفها كـ **public**. و لا يمكن تعريف الدوال فيه كـ **prviate** أو **protected** أو **final** أو **static**.

```
interface MyInterface {

    void myMethod();      // الدالة myMethod() تعتبر معرفة كـ public abstract حتى و لم نعرفها كذلك

    int myResult();       // الدالة myResult() تعتبر معرفة كـ public abstract حتى و لم نعرفها كذلك

}
```

شروط الربط بين الكلاس و الإنترفيس

- لا يمكن إنشاء كائن من إنترفيس.
- يستطيع الكلاس أن يرث من كلاس واحد، أي يستطيع أن يفعل **extends** لكلاس واحد.
- لا يستطيع الكلاس أن يرث من إنترفيس، أي لا يستطيع أن يفعل **extends** لإنترفيس.
- يستطيع الكلاس تنفيذ إنترفيس أو أكثر، أي يستطيع أن يفعل **implements** لإنترفيس أو أكثر.
- الكلاس الذي ينفذ إنترفيس، يجب أن يفعل **Override** لجميع الدوال التي ورثها من هذا الإنترفيس
- يستطيع الإنترفيس أن يرث من إنترفيس أو أكثر. أي يستطيع الإنترفيس أن يفعل **extends** لإنترفيس أو أكثر.

تنفيذ الإنترفيس في جافا

- عند تنفيذ أي إنترفيس، يجب أن تفعل **Override** لجميع الدوال الموجودة فيه، و يجب تعريفهم كـ **public** حتى يستطيع أي كائن من هذا الكلاس أن يستخدمهم.

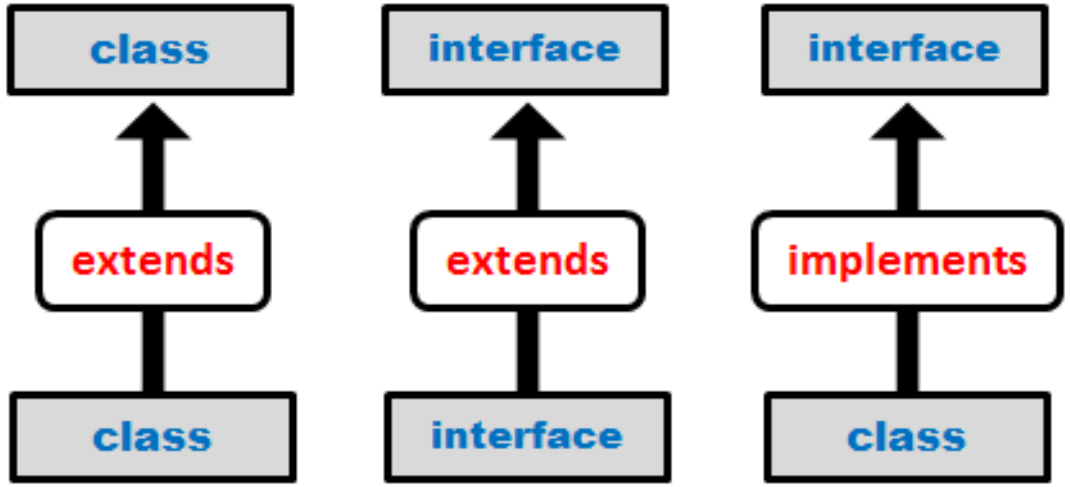
```
interface A {              // أي كلاس ينفذ هذا الإنترفيس يجب أن يفعل Override لجميع الدوال الموجودة فيه
    void print();
}
class B implements A {     // هنا قلنا أن الكلاس B ينفذ الإنترفيس A
    @Override              // الكلاس B مجبور أن يفعل Override للدالة print(). لا تنسى إضافة كلمة public
    public void print() {
        System.out.println("B should Override this method");
    }
}
```

شروط أساسية عند إنشاء إنترفيس

- لا تستخدم أي **Access Modifer** عند تعريف إنترفيس.
- لا تستخدم أي **Access Modifer** عند تعريف دالة بداخل إنترفيس.
- بداخل الإنترفيس جميع الدوال يجب أن لا تملك **body** .
- لا يمكن للإنترفيس أن يملك (constructor)
- **اسباب استخدام الإنترفيس**
- إذا كنت تنوي بناء كلاس يطبق مفهوم الـ **Full Abstraction**.
- لحل مشكلة تعدد الوراثة.
- لتطبيق مبدأ تعدد الأشكال .(**Polymorphism**)
- تعمل الإنترفيس على تعزيز إمكانية إعادة استخدام الكود (**reusability**)وقابلية صيانتها من خلال فرض مجموعة مشتركة من السلوكيات عبر فئات متعددة.
- إنها تتيح اقترانًا فضفاضًا (**loose coupling**) بين الكلاسات، مما يسمح بالمرونة وسهولة تنفيذ أنماط التصميم.

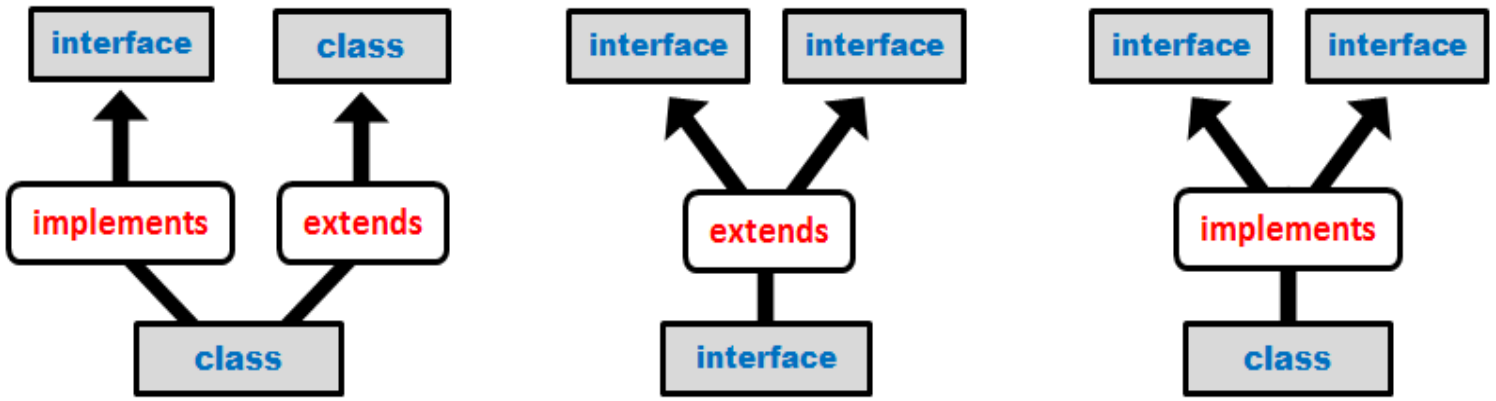
الفرق بين الكلمتين extends و implements في جافا

- **Extends** تعني وراثة، نستخدمها لجعل كلاس يرث من كلاس، أو لجعل إنترفيس يرث من إنترفيس أو أكثر.
- **Implements** تعني تنفيذ، نستخدمها لجعل كلاس ينفذ إنترفيس أو أكثر.
- الصورة التالية توضح لك متى يمكن إستخدام الكلمتين **extends** و **implements**.



أشكال تعدد الوراثة في جافا

- إذا قام كلاس بتنفيذ أكثر من إنترفيس، أو إذا قام إنترفيس بوراثة أكثر من إنترفيس، تسمى هذه العمليات وراثة متعددة (**Multiple Inheritance**).
- الصورة التالية توضح لك شكل الوراثة المتعددة.



أشكال تعدد الوراثة في جافا..

- إذا كان الكلاس ينفذ أكثر من إنترفيس، يجب وضع فاصلة بينهم.

في حال كان الكلاس يرث من كلاس آخر و ينفذ إنترفيس أو أكثر، إفعل extends للكلاس في البداية ثم إفعل implements لأي إنترفيس تريد .

- قمنا بتعريف إثنين إنترفيس **A** و **B**، و قمنا بتعريف كلاس إسمه **C** و قلنا أن **C** ينفذ **A** و **B**.

```
interface A { }
interface B { }
class C implements A, B { }
```

- قمنا بتعريف ثلاثة إنترفيس **A**، **B** و **C**. و قلنا أن **C** يرث من **A** و **B**.

```
interface A { }
interface B { }
interface C extends A, B { }     // هنا الإنترفيس C يرث من الإنترفيس A و الإنترفيس B
```

أشكال تعدد الوراثة في جافا...

- قمنا بتعريف إنترفيس **A**، كلاس **B** و كلاس **C** يرث من **B** و ينفذ **A**.

```
interface A { }
class B { }
class C extends B implements A { }     // هنا الكلاس C يرث من الكلاس B و ينفذ الإنترفيس A
```

أشكال تعدد الوراثة ف

- في المثال التالي قمنا بتعريف إنترفيس **A** يملك دالة إسمها **printA()**.
و إنترفيس **B** يرث من الإنترفيس **A** و يملك دالة إسمها **printB()**.
- ثم قمنا بتعريف كلاس إسمه **C** يطبق الإنترفيس **B**، و بالتالي عليه أن يفعل **Override** لجميع الدوال التي ورثها.

```
public interface A {
    void printA();
}
public interface B extends A {
     void printB();
}
public class C implements B {       // printB() و printA() للدالتين Override إذاً يجب أن يفعل .B و الذي
هنا الكلاس C يطبق من الإنترفيس B بدوره يرث من الإنترفيس

    @Override                // printA() للدالة Override الكلاس C مجبور أن يفعل
   public void printA() {
      System.out.println("C should Override the method printA()");
   }
    @Override                // printB() للدالة Override الكلاس C مجبور أن يفعل
   public void printB() {
      System.out.println("C should Override the method printB()");
   }
}
```

مفهوم Nested Interfaces في جافا

- يمكنك تعريف إنترفيس بداخل إنترفيس بداخل إنترفيس إلخ.. ويمكنك تنفيذ الإنترفيس الذي تريده منهم بالتحديد متى شئت.
- قمنا بتعريف إنترفيس **A**، و بداخله إنترفيس **B**، و بداخله إنترفيس **C**.
ثم قمنا بتعريف كلاس **D** ينفذ **A**، و كلاس **E** ينفذ **B**، و كلاس **F** ينفذ **C**.

```
interface A {                   // A       نكتب A لنصل للإنترفيس
    interface B {               // A.B     نكتب A لنصل للإنترفيس B الموجود بداخل الإنترفيس
        interface C {           // A.B.C   نكتب A لنصل للإنترفيس B الموجود بداخل الإنترفيس C الموجود بداخل الإنترفيس
        }
    }
}
class D implements A { }        // A هنا الكلاس D ينفذ الإنترفيس
class E implements A.B { }      // B هنا الكلاس E ينفذ الإنترفيس
class F implements A.B.C { }    // C هنا الكلاس F ينفذ الإنترفيس
```

مفهوم Nested Interfaces في جافا ..

- يمكنك أن تفعل **import** للإنترفيس و عندها يمكنك أن تكتب إسمه فقط للوصول إليه.

- هنا سنفعل **import** للإنترفيس **B** و **import** للإنترفيس **C** بدل أن نصل إليهم من الإنترفيس **A**.

```
import A.B;                 // هنا قمنا بتحديد المكان الموجود فيه الإنترفيس B لذلك أصبح يمكننا الوصول إليه مباشرةً
import A.B.C;               // هنا قمنا بتحديد المكان الموجود فيه الإنترفيس C لذلك أصبح يمكننا الوصول إليه مباشرةً
interface A {               // A نكتب A لنصل للإنترفيس
    interface B {           // B نكتب A لنصل للإنترفيس B الموجود بداخل الإنترفيس
        interface C {       // C نكتب A لنصل للإنترفيس C الموجود بداخل الإنترفيس B الموجود بداخل الإنترفيس
        }
    }
}
class D implements A { }    // A هنا الكلاس D ينفذ الإنترفيس
class E implements B { }    // B هنا الكلاس E ينفذ الإنترفيس
class F implements C { }    // C هنا الكلاس F ينفذ الإنترفيس
```

Example for interface مثال على الانترفيس

```
public class javainterf {
 public static void main(String[] args) {
       Dev3 d = new Dev3();
       d.account();
        d.loan();
        d.deposit();
        d.withdraw();
    }
}
// Level 1
interface Bank {
    void deposit();
    void withdraw();
    void loan();
    void account();
}
// Level 2
 class Dev1 implements Bank {
    public void deposit() {
        System.out.println("Your deposit Amount :" + 100);
    }
}
 class Dev2 extends Dev1 {
    public void withdraw() {
        System.out.println("Your withdraw Amount :" + 50);
    }
}
```

```
Your deposit Amount :100

Your withdraw Amount :50
```

```
// Level 3
class Dev3 extends Dev2 {
    public void loan() {
    }
    public void account() {
    }
}
```

Thank you for good listening....

Any Questions !