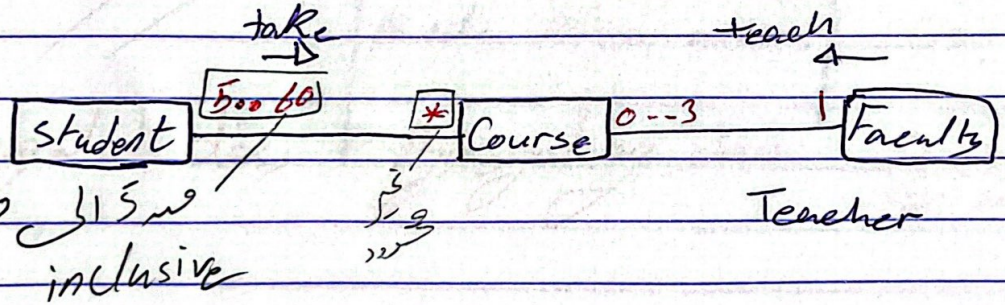
Class RelationShips

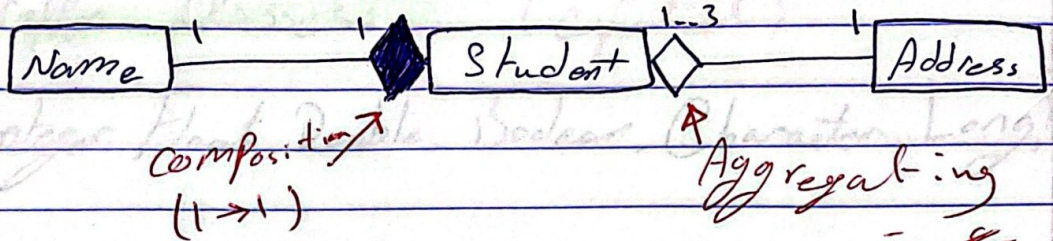
1 ⊛ Association:



Public class Student Private Course[] courseList; public void addCourse	Public class Course private Student[] classList; public void addStud public void setFaculty	Public class Faculty private Course[] courseList; public void addCourse

⊛ هذه اضيف في

2 * Aggregation and Composition:
(ownerShip) have a ‹ has a · علاقة ملكية



Aggregating class/object: Student/s1

Aggregated class/object: Name/ahmed

Comosition is Exuclusive. (علاقة حصرية)

Public class Name	Public class Student	public class Address
//	private Name name;	//
	private Address address;	
Aggregated class	Aggregating class	Aggregated class

Inheritance and Polymorphism.

* inherence is useful for code-reusing.

Parent class : base class : super class

Child class : extended class : ~~extended~~ : derived class : sub class

by default : the parent is Object class as long as i didn't determine the Parent

in UML

* inheritance is is-a relation Ship

* inheritance is single inheritance
(interfaces الوراثة فقط من أب واحد (عدا الكلمة المحجوزة

* public class Worker extends Person {

// super Keyword
- Call superclass constructor (#First line)
- Call superclass method.

super : لينادي ~ ~ الخاص بأبيه الثاني داخل الكلاس نفسه constructor لينادي ال

كلاهما لازم يكونه اول سطر عند نداءه constructor

← بالتالي مستحيل ان يكونا مكتوبين معا

ex

```
Public class Person{
String name;
Private int id;
Public Person(String name, int id){
    this.name = name;
    setId(id);
}
    int getId(){
        return id;
    }
    Void setId(int id){
        this.id = id;
    }

Void printHello(){
    __.println("hello");
}
```

```
Public class Worker extends Person{
    double salary;
    public Worker(int id, string name, double salary){
    // this.id = id;
```

ممنوع لانه private

اضافية

طريقة (1)

```
    setId(id);
    this.name = name;
    this.salary = salary;
```

اخذ كل الصفات اللي فيه

طريقة (2)

الاستعمال الاول super

```
    Super(name, id);
    this.Salary = salary;
```

الاستعمال الثاني super

```
    Super.printHello();
}
```

الاستعمال الثاني لاستدعاء فنكشن (بنربط عشان property).

* احيانا بنربط انادي الفنكشن بدون super ، ليس دائما

super.name = name; ✓ this.name = name; ✓

متى يلزم استدعاء super ؟؟ method

- عندما يكون للاب والابن method بنفس الاسم والاكم نفسه.
- لو نادينا دون super ← اقصر تبعية الابن
- مع super ← تبعية الاب.

---

ex

```
Public class A {
(1) Public A () {
      this (" this is A no-arg construction");
    }
(2) Public A (String str) {
      system.out. Println(" this is A arg cons..");
    }

Public class B extends A {
(3) Public B () { ... Print(" swapping "); }
(4) Public B (string st) {
      System ... Println (" this is B arg ...");
    }

Public class C extends B {
(5) Public C () {
      Super (" this is C no-arg ...");
      ... Println (" this is the end of C no arg");
    }
(6) public C (string s) {
      println(s);
    }

Public static Void main (String [] args) {
    C c = new C (" hello");
}
```

آلية العمل

١- جاء على C أخذ رقم 6.
٢- يدخل يبحث عند اول جملة ⟵ اذا this ينتقل على Constructor آخر مناسب داخل الكلاس نفسه ⟵ اذا لم يجد
٣- يدخل يبحث عند super ⟵ لينتقل الى constructor مناسب في الأب ⟵ إذا لم يجد ⟵.
٤- اذا لم يتوفر كلاهما
⟵ يضع super() وينتقل على no-args const. الذي في الأب.
٥ (*) لذلك ضروري حينها ان يكون موجود بالاب واحد فاضي
يكون هنا قبل تنفيذ اول جملة مكتوبة

٥- يرجع بالتدريج وينفذ ما هو مكتوب في كل Constructor مرّ خلاله (رجوعاً)

output → this is A args const.
swapping
hello

Q. نفس السؤال ولكن غيّر main
C c = new C();

output → this is A args cons.
this is B args cons.
this is the end ~~of~~ of C no-arg.

(*) إذا A لم نكتب فيه super / this.
نعمل super ~~...~~ وينتقل إلى object
ثم لا ينفذ فيه شيئاً

Note to avoid errors

⇒ always write no-arg constructor in super class

// Overriding methods

* overloading ← لا يهتم اختلاف الـ return type
← المهم ان يختلف الـ args او ترتيبهم
← نفس الاسم.
← يتواجد في الكلاس الواحد او بين الاب وابنه

signature تختلف

لا يختلف الاسم

Signature: الاسم والنوع وعدد الـ args وترتيبهم

overriding:

exactly the same Signature

* لازم يكونوا بين الكلاس وابنه
* دائما يرجع نفس النوع.

Note @ override

Use it before you write the overrided method to check if you Wrote it well.

overriding Vs Overloading

```
in main :
    A a = new A();
    a.p(10);
    a.p(10.0);
    a.q(10);
```

(1) (2)

```
class B {
public void p(double i){
   print(i*2);

Public void q(double i){
   print(i*3);
}
```

```
class B {
public void p(double i){
   print(i*2);
}

Public void q(double i){
   print(i*3);
}
```

```
public class A
   extends B {
public void p(double i){
   print(i);
  }
}
```

```
class A extends B {

Public void p(int i){
   println(i);
```

overriding output

10.0
10.0
30

اضغط
طباعة يطبّق على الابن
لا ينقل للاب

output

10 int
20.0
30.0

overloading

overriding: يأتي الأول
overloading: يأتي الأدنى

(3)

```
class B {
    void p(int i) {
        print(i*2);
    }

    void q(double i) {
        print(i*3);
    }
}

class A extends B {
    void p(double i) {
        println(i);
    }
```

output

حتى لو انه int لكن وضعه في p داخل الابن

⇐ الاولوية بـ overloading ⇐ ينقل التي في الأب

```
20 int
10.0
30.0
```

* اذا كنت بدي التي في الأب (دون ترك الأمر لو ان مختار)

⇐ اقوم super لاذهب للأب

toString → ~~prints~~ returns all Properties

```java
public class Person {

    String name = "ahmad";
    int id = 111;
    public String toString() {
    return "name = " + name + " id = " + id;


}
  public class Worker extends Person {
    double salary = 1000.5;

    Public String toString() {
  return ~~Supp~~ super.toString() + "Salary = " + salary;
   }


}
   System.out.println(W.toString());
```

output → name = ahmad id=111 salary= 1000.5

```java
   System.out.println(W);
```

output → نفسه

∴ by default ⇒ it calls toString
→ So it's important to implement toString in each class

*/ دائما اعمل toString في كل class

it ~~prints~~ returns all properties ⇐

* مهم لا تطبعو ← [illegible]

مثلاً Person P = new Person();

___print(P);

output → calls toString in Person

→ name="ahmad" id=111

لو لا يوجد في Person toString

System.out.println(p);

→ class name @ ١١١

يطبع اسم الكلاس وموقعه

لو لا يوجد toString في Worker

___print(w);

→ name = ahmad id=111 يطبع قيمة الاب

لو لا يوجد toString في الأب ~~الابن~~

ويوجد في الابن

~~يطبع موقع الابن~~

 Print(w)

يطبع موقع (الاب او الابن) , يطبع العضاء في W الخاصة فيه فقط

* الحل على مزاجنا لاكتاب super وليس على مزاجك طباعة كل property على حدة.

* عندما أرث من الأب وأريد التعديل على سلوك معينة
⇐ اعمل override في كلاس الابن.

// Polymorphism (many forms)

* أستطيع أن أمرر الـ subtype او الـ sub/subtype في مكان يتوقع فيه أن أدخل supertype.

declared type Vs actual type

Person p = new Worker();
declared type ↑ ↳ actual type

* عندما يعمل Build (compile) يفحص الـ declared ⇐ يظهر error او لا يظهر بناءً على النوع الظاهر.

* عندما ينفذ (Run): يعتمد على الـ actual type

Dynamic binding: (يعتمد على أساس الـ actual type)

Suppose We have these classes.

class A extends Object { فيه toString() }
class B extends A { فيه printVal() }
class C extends B { فيه toString }
class D extends C { فيه printVal() }

---

class Driver {

Void showObject(Object O) {
    print(O.toString());
}

public static void main() {

Object O = new B();
showObject(O);

→ آلية الحل:
1/ يعين الـ ~~actual~~ declared ← Object و فيه toString ← errors لا
2/ يروح الـ actual ← واذا لا فيه الـ method المذكورة، يبحث عنها عند الآباء (يأخذ عند اقرب أب فيها)

output: يطبع الخارج ← A

أو
Object O = new D()
showObject(O);

output: يطبع الخارج ← toString ← C

أو
Object O = new C();
showObject(O)

output → يطبع الخارج ← toString ← C

* يجب ان يكون الـ declared هو الـ ~~Sub~~ actual type او أحد ابناء الـ declared ، لكن ليس أباه ..

* لا يمكن ان اعرّف احد الاباء لخانة الـ actual type
لكن " " " " الابناء " " " " "

فرضاً

show Object (B b) → declared
يقبل من نوع B ، C فقط

show Object (A a)
يقبل declared A, B, C

show Object (O o)
يقبل O, A, B, C في declared

الخلاصة : الابن يدخل مكان أبيه
الأب لا يدخل مكان الابن بناءً على declared

حالات أخرى للمثال السابق

حالة

```
Public Void showObject (C c){
    print (c.printVal());
```

~~in main~~ هل يوجد error؟

لا ، يفحص هل C او احد ابائه يحتوي printVal ← نعم

حالة

```
Public ___ showObject(B b){
___ Print(b.printVal()); }
Object o = new C();
showObject(o);
```

هل يوجد errors؟

1) يتفحص عند B ← هل هو او ابائه يحوي printVal ← نعم

2) الاوبجكت o ظاهريا هو من نوع Object
← لا يمكن ارسالها بناءً على الـ actual
لانه O ظاهريا هو أب لـ B

⇒ error

* Casting Objects (upcasting / downcasting) and instanceof

Suppose I have to classes.
Person, and subclass : Worker

```
in Person : int getId(){ return id;}
in Worker : int getId(){return super.getId() + 5 ;}   ← override  أصل
in Driver :
  public static void printObject (Person p){
      print(p.getId());
  }

in main :
  printObject (new Person (11 , "ahmed"));
```

output → 11

```
printObject(new Worker(14, "Ali", 1500));
```

actual type 's Person ابن ارسلت ~~~ ← Dynamic binding

output → {19} ⇐ يذهب على المعرّفة ← للبعي

لو ← showObject ← كان فيه ~~printSalary()~~ p.getSalary()
← error ← لأنه لا يوجد هذه الكاطرية في Person

* ولكن في الحالة ان احصل ال method
هي عامة ، ترابط للاب ، لمكن استخدامها
مع كل الأبناء

∴ ∴ I need Casting

لكن ، لو عملت Casting وأدخلت الأب ← سيسبب مشاكل.
لانه أجبرته ان يعتبر Person انه Worker وعندما مررت Person.
← ان يجد getSalary

(Downcasting)
← الحل نستعمل instanceof + Casting
← للتأكد انه فعلاً من نوع الابن لنصحح ال Casting

```
public static void printObject(Person p){
    if (p instanceof Worker)
        print("id" + p.getId() + "salary" + ((Worker)p).getSalary());
```

يفحص هل p له actual type
من نوع Worker او احد ابناء Worker
(لانه الابن يدخل مكان الأب)

```
    else "id=" + p.getId(); }
```

in main
showObject(new Person(10, "Ali"));
output → ~~No output~~ 10

ShowObject(new Worker(11, "Ali", 1500));
→ id = 11   Salary = 1500
← اهلا انه عملت override واضفت 5 على id

ممكن أستعمل else بعد if واطبع id وحدها

SO: DownCasting:

Person p = new Worker();
Worker w = p;
→ error   خاطئة وضعت الاب مكان الابن

⇒ Worker w = (Worker) p; ✓

So: up casting:

it is implicit.

→ Worker w = new Worker();
Person p = w;
→ ✓ وضعت الابن مكان الاب

Person p = (Person) w;
→ also true ✓

Object equals method.

لازم هكذا

```
Public boolean equals (Object o){
    return (this == o);}
```

true ← اذا يؤشران على المكان نفسه ✗

لبس عالذي في String او Date ←
عمل لهن override

Q لو قررت اعمل equals في Person ← وقررت ان التساوي هو ال id

→

```
@override
Public static
Public boolean equals (Object o){
    if( o instance of person)
        return id == ((Person) o).getId();
}
else
    return this == o;
```

لازم هكذا حتى تحصل ع override

ممكن ادخل worker لانه الوه Person ← (if)

اذا مش نفس ال id
على الاقل افحص اذا يؤشر على المكان نفسه

التنبيه: (int, double) Simple type لـ ArrayList يمنع استخدام

الحل: ArrayList<Integer> list = new Arraylist<> ();

```
list. add(new Integer ("4"));      Wrapper type
list. add (new Integer(6));
list. add( 8) ;

System. out. println( ~~list. max(~~ Collections. sort(list) );
    -> 8
```

لكن هذا غير محبب استخدامه.

البحث: Static Method لـ override هل يمكن عمل Jan, 2nd محاضرة

// Protected Modifier

subclass can invoke from any ~~bac~~ package.

	from the same class	from same package	Subclass in any package	from any package
public	✓	✓	✓	✓
protected	✓	✓	✓	
default	✓	✓		
~~protected~~ → Private	✓			

Protected in UML: #

جرب هل يمكن الوصول على protected من main

Final class and methods and Vars

public final class Name {}

- لا يمكن إنشاء subclass له
- لا يورث

public final void printInfo() { ... }

- لا يمكن عمل override
- عادي يرث الـ method لكنه لا يمكنه عمل override
- لو اختلفت Signature أخر ← لا يكون override ← يجوز

final int NAME = 4;

constant