قسم الهندسة المعلوماتية

برمجة 3

Java Programming

ا. د. علي عمران سليمان

محاضرات الأسبوع الثالث

inherited class

الفصل الأول 2024-2025

Contents 1

1. Objects and Classes .	11. Inheritance and Constructors.
2. UML class diagrams .	12. Overriding Superclass Methods.
3. Performing output Displaying with print, println, printf.	12.1. Reference type and object type
4. Performing Input Scanner and some of its methods.	3.6 Class JOptionPane Using Dialog Boxes
5. default constructor.	showMessageDialog(), showInputDialog()
6. Overloaded Constructors, and methods.	4.15 GUI &Graphics,
7. Static Method , and Data fields.	4.15 Creating Simple Drawings—Displaying
8. Call by value and references.	and drawing lines on the screen
9. copy constructor.	5.11 Drawing Rectangles and Ovals—Using
10. **inherited class**.	shapes to represent data.

References

- Deitel & Deitel, Java How to Program, Pearson; 10th Ed(2015)
- د.علي سليمان، بنى معطيات بلغة JAVA، جامعة تشرين 2013-2014

جامعة المنارة
MANARA UNIVERSITY

What is Inheritance?

- الكائنات الو اقعية هي عادةً نسخ متخصصة من كائنات أخرى أكثر عمومية.

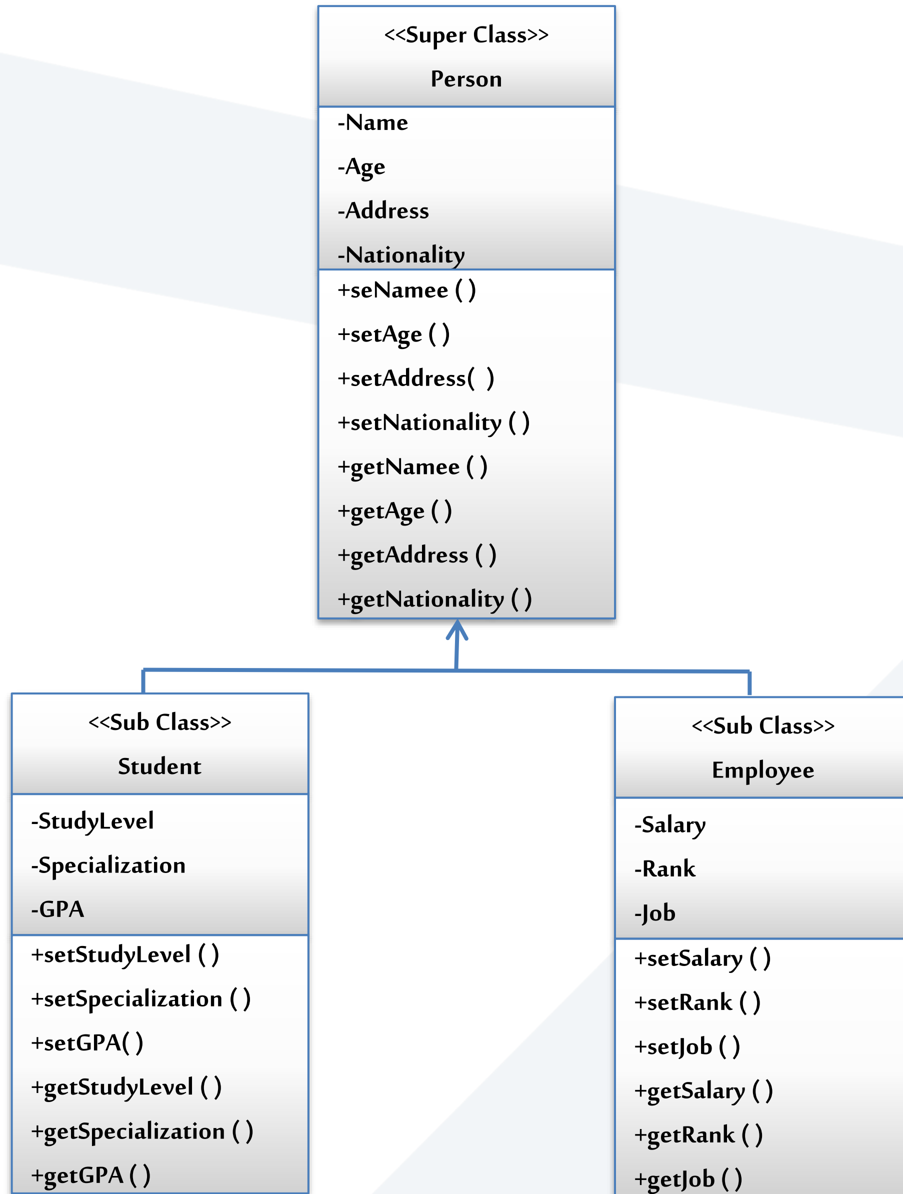
- يصف مصطلح "الطالب" Student نوعًا عامًا جدًا من الطلاب ذوي الخصائص المعروفة، وهو بدورة نوع خاص من الشخص Person وكلاهما له ميزات مشتركه كمعطيات مثل: اسم، وعنوان، وجنسية ... وكطرق مثل setter و getter لهذه المعطيات.

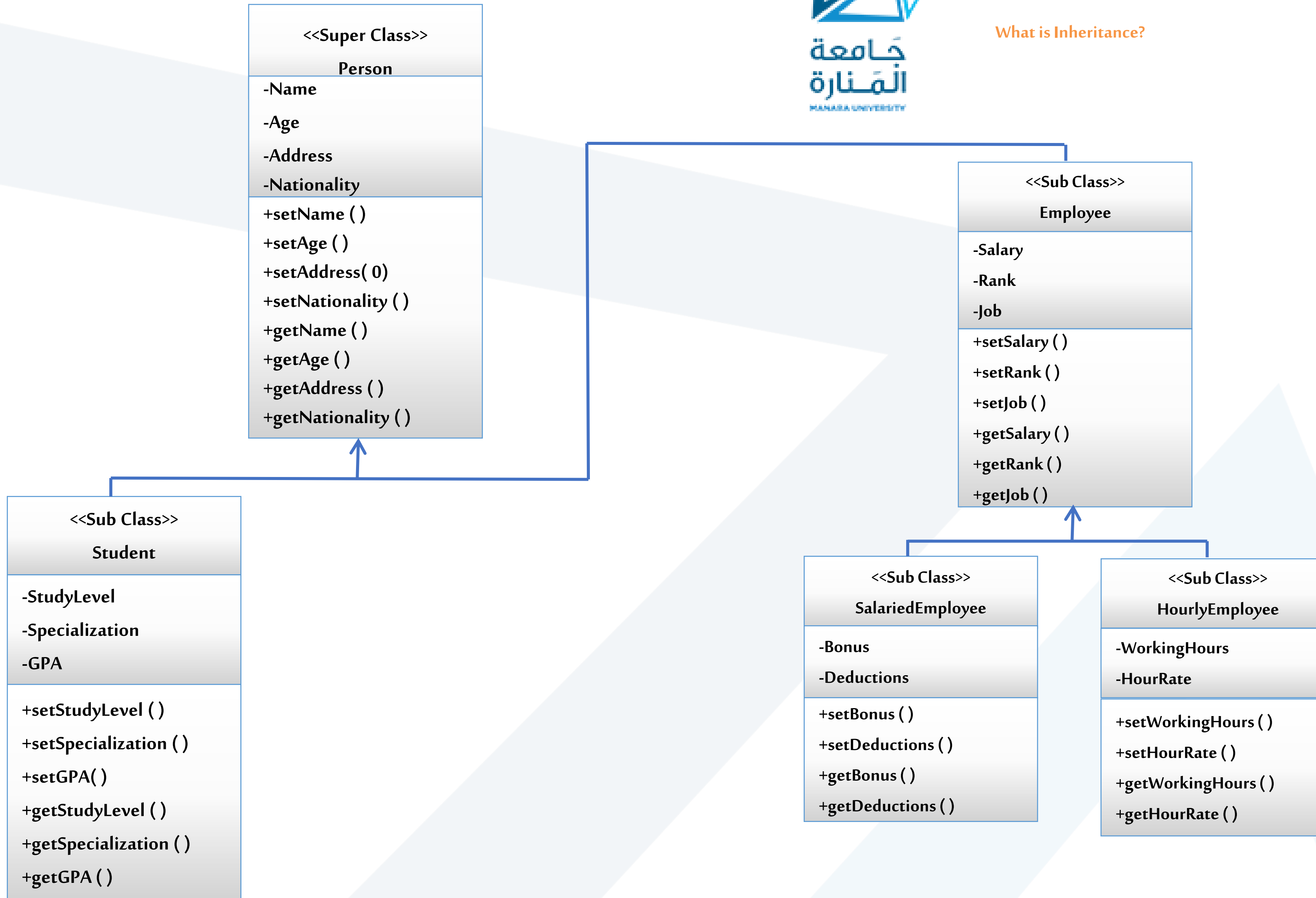
- طلاب الدراسات العليا Post Graduated والطلاب الجامعيين Under Graduated هم نسخ متخصصة من الطلاب.

  - يتشاركون في الخصائص العامة للطالب مثل الاختصاص Specialization والمعدل GPA.
  - ومع ذلك ، لديهم خصائص خاصة بهم.
    - بعد التخرج مجال البحث الذي يهتمون به، المستوى ماجستير أو دكتوراه.
    - قبل التخرج لديهم رقم مجموعة و رقم صف.

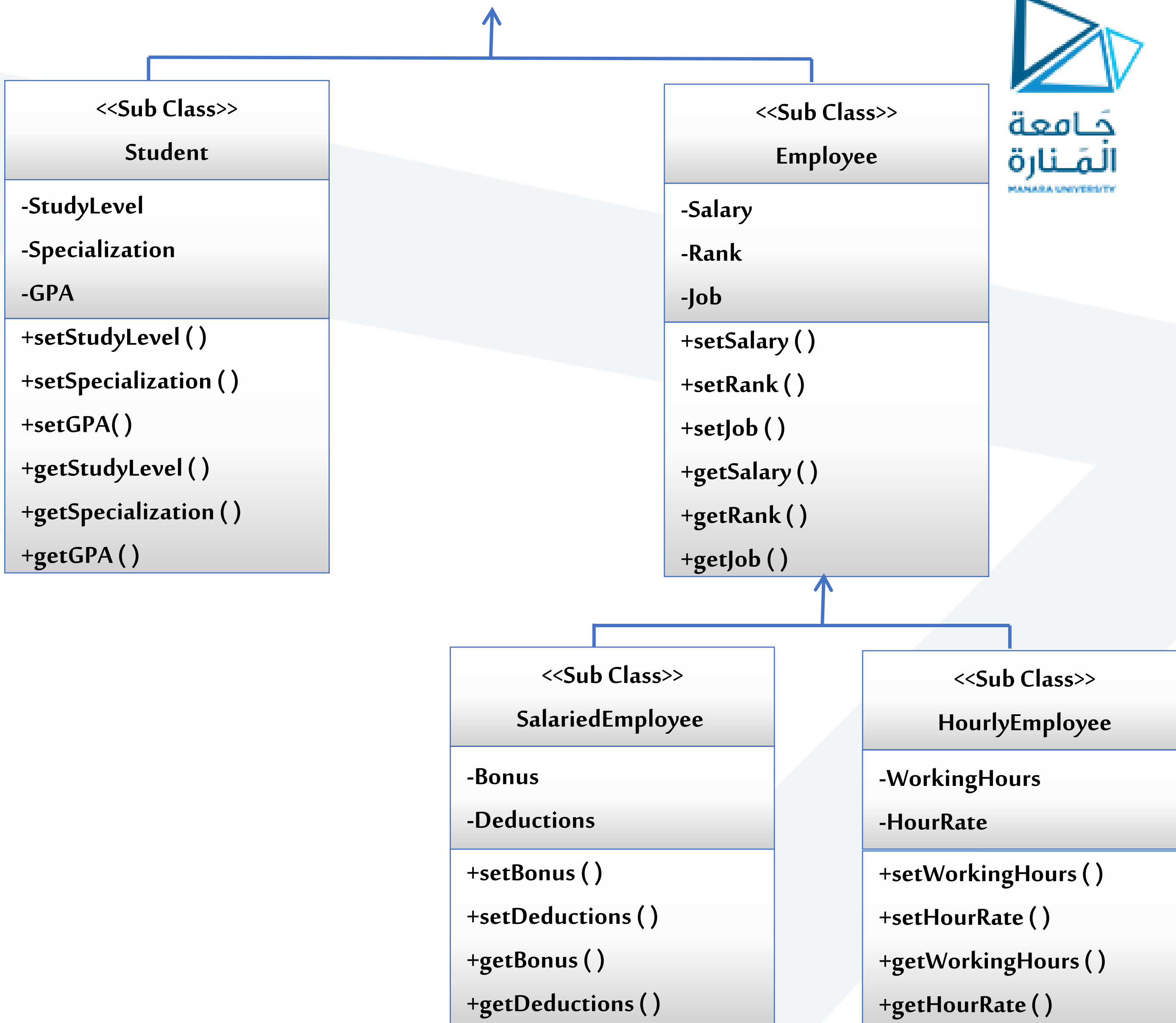
What is Inheritance?

Generalization vs. Specialization




<<Super Class>>
Person
-Name
-Age
-Address
-Nationality
+setName ( )
+setAge ( )
+setAddress(0)
+setNationality ( )
+getName ( )
+getAge ( )
+getAddress ( )
+getNationality ( )
<<Sub Class>>
Student
-StudyLevel
-Specialization
-GPA
+setStudyLevel ( )
+setSpecialization ( )
+setGPA( )
+getStudyLevel ( )
+getSpecialization ( )
+getGPA ( )
<<Sub Class>>
Employee
-Salary
-Rank
-Job
+setSalary ( )
+setRank ( )
+setJob ( )
+getSalary ( )
+getRank ( )
+getJob ( )
<<Sub Class>>
SalariedEmployee
-Bonus
-Deductions
+setBonus ( )
+setDeductions ( )
+getBonus ( )
+getDeductions ( )
<<Sub Class>>
HourlyEmployee
-WorkingHours
-HourRate
+setWorkingHours ( )
+setHourRate ( )
+getWorkingHours ( )
+getHourRate ( )

What is Inheritance?



The “is a” Relationship

- العلاقة بين الطبقة العليا superclass المورثة والطبقة الوارثة inherited class تسمى علاقة " is a ".
  - طالب الدراسات العليا " is a " طالب Student.
  - الموظف " is a " شخص Person.
  - الموظف بأجر " is a " موظف Employee.
  - السيارة " is a " مركبة vehicle.
- الكائن المخصص تملك:
  - جميع خصائص الكائن العام وخصائص إضافية تجعلها مميزة لها أو إعادة كتابة بعض الطرق بدون إضافة خصائص.
- في البرمجة OOP، يتم استخدام الوراثة لإنشاء علاقة " is a " بين الفئات وما تعرف بالوراثة inheritance .

ملاحظة1: إذا أضاف الصنف الوارث معطيات data members أو طرق methods إلى ماتمت وراثة يوصف is like a.

ملاحظة2: عند استخدام صنف عدة أصناف و أنشاء كائن منه فيعبرعن تركيب هذه الأصناف ضمنه ويعرف بـ has a مثل:

-car has a door, car has a window (Composition)

The “is a” Relationship

- يمكننا توسيع extend قدرات الصنف.
- **Inheritance يشمل الطبقة العليا superclass والطبقة الفرعية subclass.**
  - **صنف الأب هو الصنف العام أو المعمم general.**
  - **الصنف الفرعي هي الصنف المخصص specialized.**

- **The subclass is based on, or extended from, the superclass.**

- **تسمى الطبقات المورثة بطبقة الأباء** ParentClasses أو العليا Superclasses **أو الأساسية** Base**Classes** ،
- **تسمى الطبقات الوارثة بالأبناء ChildClasses، الفرعية** subClasses أيضًا **الطبقات المشتقة** derived**Classes**.

- **يمكن اعتبار العلاقة بين الأصناف بمثابة صنف للوالدين واصناف للأبناء as parent classes and child classes.**
- **ترث الاصناف الفرعية subclass (الوارثه) الحقول والأساليب من الاصناف المورثه superclass دون إعادة كتابة أي منها.**
- **يمكن أن تضاف حقول وأساليب جديدة إلى الصنف الفرعي subclass.**

Inheritance

- يتم استخدام الكلمة المفتاحية extends في Java، في سطررأس الصنف مابعد الصنف الوارث وقبل الصنف المورث.

public class Employee extends Person

- كما هو معروف: - نستفيد في مبدأ إعادة الاستخدام Reusability بدون كتابة المشترك بين الأصناف مرتين.
  - عند إضافة حقل معطيات أو طريقة في صنف الأب مرة واحدة ستظهر عند كل الورثة.
  - يمكن التعديل على الطريقة التي نحتاج التعديل عليها ماعدا الطرق final.
- أعضاء الصنف المورثه superclass التي تم تعريفها على أنها خاصة:

- لاتورث. - موجودة في الذاكرة عند إنشاء كائن من الصنف الفرعي subclass ولا يمكن الوصول إلى الأعضاء الخاصة للصنف المورث من قبل الصنف الفرعي subclass إلا بالطرق العامة public methods للصنف الأعلى superclass.

- أعضاء الصنف superclass المعرفين public or protected:

- موروثة من قبل الصنف الفرعي.- يمكن الوصول إليها مباشرة من الصنف الفرعي subclass.

Inheritance, Fields and Methods

- When an instance of the subclass is created, the non-private methods of the superclass are available through the subclass object.

```
Employee emp1 = new Employee();
 emp1.set_Age(30);
System.out.println("Age = " + emp1.get_Age());
```

- Non-private fields of the superclass are available in the subclass.

```
Set_Age(30);
```

جَامعة المَنارة
MANARA UNIVERSITY

- المنهج الباني لايورث أي لايمكن تعديل باني الصنف الأساس من الصنف المشتق، بل يمكن استدعائه.

- عندما يتم إنشاء مثيل لصنف فرعي ، يتم تنفيذ الباني الافتراضي لصنف الأب **superclass** أولاً وببساطه يمكن وضع رسالة إخراج في باني **Person** الافتراضي تميزه، ورساله إخراج في باني الصنف **Student** المشتق تميزه و انشاء كائن من الصنف **Student** سيتم طابعة رسالة الصنف الاب تم رسالة الصنف الابن.

- يمكن استدعاء باني الصنف الأب بشكل صريح من الصنف الأبن باستخدام الكلمة المحجوزة **.super** والجد **.super .super** والاصل أن يتعامل الابن مع الاب فقط.

- إذا تم تعريف الباني مع البارامترات في صنف الأب .
  - يجب أن يوفر صنف الأب باني بدون بارامترات **.no-arg** أو
  - يجب أن يوفر الصنف المشتق باني ويجب أن ينادي باني الأب.

- يجب أن تكون الاستدعاءات لباني الأب هي أول عبارة جافا في باني الابن.

جامعة المنارة
MANARA UNIVERSITY

Overriding Superclass Methods

- قد يكون للصنف الفرعي طريقة لها نفس التوقيع مثل طريقة الصنف الأب.
  - ✓ إذا اختلفت طريقة الصنف الفرعي في التنفيذ عن طريقة الصنف الأب وجبت إعادة كتابتها overrides بما يناسبها.
  - ✓ يُعرف هذا باسم إعادة كتابة overrides وبالتالي ستغطي الطريقة الجديدة الطريقة السابقة(الموجوده في الاب).
- لدينا طريقة موجودة ضمن الصنف Employee اسمها getSalary() وضمن الصنفين المشتقين وبنفس التوقيع إنما بمعادلة حساب مختلفه (صنف الابEmployee يملك راتب فقط، الصنف المشتق SalariedEmployee لديه راتب وله مكافأت وعليه خصومات، الصنف المشتق HourlyEmployee يعمل بالساعة عدد من الساعات وللساعة سعر) .
- ضمن الصنف Employee تحسب getSalary() {return salary;}
- ضمن الصنف المشتق SalariedEmployee بطريقة حساب getSalary() {return salary+bonus-deductions ;}.
- وضمن الصنف HourlyEmployee بطريقة حساب مختلف getSalary(){return working_hours*hours_rate;}
- يستدعي كائن من الصنف الفرعي نسخة الطريقه للصنف الفرعي الخاصة به، وليس طريقة صنف الأب.
- يحبب استخدام التعليق التوضيحي @Override قبل إعلان طريقة الصنف الفرعي مباشرةً وتؤمن الانتقال السهل مابين المنهج الأصل والمنهج الذي عمل لها override في بعض بيئات برمجة Java.

Calling The Superclass Constructor

Overriding Superclass Methods

Employee

```
public double get_salary( )
{
return salary;
}
```

Salaried Employee

```
public double get_salary( )
  {
    return salary+bonus-deductions ;
  }
```

Hourly Employee

```
          public double get_salary( )
  {    return working_hours*hours_rate;
  }
```

جامعة المنارة
MANARA UNIVERSITY

Overriding Superclass Methods

- عند إجراء **overridden** لطريقة في صنف الابن موروثة من صنف الأب، يمكن استخدام الكلمة المفتاحية super من اجل نداء طريقة صنف الأب من صنف الأبن ، لنفرض لدينا منهج اسمه ()printAllDatails لطباعة كل البيانات ضمن الصنف Employee وتمت إعادة كتابته ضمن صنف SalariedEmployee وبالتالي عندما نرغب بطباعة بيانات كائن من الصنف SalariedEmployee نحتاج مناداتها وكتابة مايخص الصنف الجديد فقط

```
public void printAllDatails()
        {super.printAllDatails();
        System.out.println("\n rank "+rank+"\n job "+job +"\n salary = " + getSalary()); }
```

- هناك يجب التمييز بين التحميل الزائد للطريقة و **overridden**.
- في التحميل الزائد كما هو معروف عندما يكون للطريقة نفس اسم طريقة أخرى أو أكثر ، ولكن بتوقيع مختلف عندها لدينا نسختين أو أكثر منها، عند **overridden** يكون لدينا نسخة واحدة أي احداها تغطي الاخرى.
- يمكن أن يحدث التحميل الزائد في علاقة الوراثة **inheritance** وغيرها.
- يمكن أن يحدث **overriding** فقط في علاقة الوراثة.

Preventing a Method from Being Overridden

- معدّل الوصول final في طريقة الصنف الأساس سيمنع إجراء overriding من قبل صنف المشتق.

```
public final double getSalary() {
        return salary;
}
```

- إذا حاول الصنف المشتق فئة فرعية override لطريقة final، يقوم المترجم بإنشاء خطأ.

- يجب التأكيد على استخدام طريقة خاصة ضمن الصنف المشتق بدلاً من نسخة معدلة من الصنف الأساس.

Protected Members

- توفر Java نوع وصول ثالث المحمية **protected**.
- يقع مفهوم وصول الاعضاء المحميين ما بين الخاص والعامة.

- استخدام النوع المحمي **protected** بدلاً من الخاص يجعل بعض المهام أسهل.
- أي صنف مشتق من صنف أخر، أو في نفس الحزمة ، لها وصول غير مقيد إلى الأعضاء المحميين.
- من الأفضل دائمًا جعل جميع حقول المعطيات خاصة ثم توفير طرق عامة للوصول إليها.

- إذا لم يتم توفير محدد وصول لعضو في الصنف، فسيتم منح عضو الصنف وصولاً على مستوى الحزمة وهو الحال الافتراضي. أي يجوز لأي طريقة في نفس الحزمة الوصول إلى هذه الأعضاء .

- أعضاء الصنف المحميين نميز حالتين:

-الوارث والمورث في نفس الحزمة package يمكن للصنف الوارث الوصول إليها وينطبق على مفهوم friendly.

- في حزمتين مختلفتين تشابه private وإنما يمكن الوصول إليها مباشرة من الصنف الفرعي **subclass**.

Protected Members

```
Package A1;
public class Shape
{
   private double height;  // To hold height.
   private double width;   //To hold width or base

   /**
    * The setValue method sets the data
    * in the height and width field.
    */

  public void setValues(double height, double width)
  {
     this.height = height;
     this.width = width;
  }
}
```

```
Package A2;

public class Rectangle extends Shape
{

   /**
    * The method returns the area
    * of rectangle.
    */

  public double getArea()
  {
   return height * width;
  //accessing protected members
  }
}
```

Protected Members

```
public class Person
{               private String   name;
                private double age;
                private String address;
                private boolean nationality;
                public Person()
                {System.out.println(" def. const Parent run \"super class\" \n "); }
                public Person(String n, double age, String ad, boolean nat)
                {System.out.println(" par. const Parent run \"super class\" \n ");
                name = n;     this.age=age;         address = ad; nationality= nat;    }
                public void setName(String n){name=n;}
                public void setAge(double a){age=a;}
                public void setAddress( String ad){address=ad;}
                public void setNationality(boolean b){nationality = b;}
                public String getName(){return name;}
                public double getAge(){return age;}
```

Protected Members

```
            public String getAddress(){return address;}
            public boolean getNationality(){return nationality;}
      public void printAllDatails(){System.out.println(" -\n name = "+name+ "\n Age "+ age +"\n Address "+ address + "\n nationality = " +nationality);}
      }// end Person

 public class Student extends  Person
 {     private int stadyLevel;
       private String specialization;
       private double GPA;
       Student(){System.out.println(" def. const Student run \"sub class\"\n");  }
       Student(String n, double age, String ad, boolean nat,int sL, String sp, double gpa){System.out.println(" par. const Student run \"sub class\"\n");
 /* The Student class has 7 parameters, 4 inherited from class Person and 3 declared For this reason we will call the constructor of the parent class and send it 4 parameters */
```

Protected Members

```java
        super( n, age, ad, nat);
        stadyLevel=sL;      specialization=sp;
        GPA=gpa;      }
    //Override method printAllDatails

@Override
    public void printAllDatails()
    {       super.printAllDatails();
            System.out.println("\n stadyLevel = "+stadyLevel +"\n specialization
                "+specialization+"\n GPA "+GPA); }

    /*public void print() { System.out.println("\n name ="+ getName()+ "\n age=
"+ getAge()+"\n address ="+ getAddress()+ "\n nationality = "+
getNationality()+"\n stadyLevel = "+stadyLevel + "\n specialization
"+specialization+"\n GPA "+GPA);  }*/
```

Protected Members

```
        public void setStadyLevel(int stadyLevel) {this.stadyLevel = stadyLevel;}
        public int getStadyLevel() {return stadyLevel; }
        public void setSpecialization(String specialization) {
        this.specialization = specialization;    }
        public String getSpecialization() {return specialization;    }
        public void setGPA(double gpa) {  GPA = gpa;    }
        public double getGPA() {return GPA;}
}// end class Student

public class Employee extends Person
{       double salary;              double rank;        String job;
        Employee(){System.out.println(" def. const Employee run \"sub class\"\n");}
        Employee(String n, double age, String ad, boolean nat, double sa,double ra,
                String jo)    {       super(n, age, ad, nat);    salary=sa;
                rank=ra;                  job=jo;       }
```

Protected Members

```
//Override method printAllDatails
@Override
public void printAllDatails()
{       super.printAllDatails();
        System.out.println("\n rank "+rank+"\n job "+job +"\n salary = "
+ getSalary());       }
//If the method signature is changed in the base class,
// Java will alert that this method has been Override
//2public double getSalary();
public  double getSalary() {return salary;}
public void setSalary(double salary) {  this.salary = salary;}
public double getRank() {return rank;    }
public void setRank(double rank) {this.rank = rank;   }
public String getJob() {return job;      }
        public void setJob(String job) {  this.job = job;}
}// end class Employee
```

جَامعة المَنارة

Protected Members

```java
      public class SalaredEmployee extends Employee
{double bonus; double deduction;
 public SalaredEmployee(){}
public SalaredEmployee(String n, double age, String ad, boolean nat, double sa,
double ra, String jo, double bo, double det)
{      super(n, age, ad, nat, sa, ra, jo);bonus=bo; deduction=det; }

@Override
public double getSalary() {return salary + bonus-deduction;}
@Override
public void printAllDatails()
{      super.printAllDatails();
       System.out.println("\n bonus = "+bonus +"\n deduction "+deduction+ "Salary
= "+ getSalary() );}
}// end class SalaredEmployee
```

Protected Members

```java
public class HourlyEmployee extends Employee
{       double houreRate;            double numberOfHours;
        public HourlyEmployee() {   }
        // TODO Auto-generated constructor stub
public HourlyEmployee(String n, double age, String ad, boolean nat,double sa,
double ra, String jo, double hR, double nOH)
{super(n, age, ad, nat, sa, ra, jo); houreRate=hR;    numberOfHours=nOH;   }
        @Override
        public void printAllDatails()
        {       super.printAllDatails();
System.out.println("\n houreRate = "+ houreRate+"\n numberOfHours"+numberOfHours +
"\n Salary = "+getSalary() );     }
        @Override
        public double getSalary() {return houreRate * numberOfHours;}

}// end class HourlyEmployee
```

Protected Members

```java
public class StudentTest
{ public static void main(String [] argc){

/*  When creating an object of a child class without parameter,
 *  it will call the constructor of the parent class
 *  and then the constructor of the child */
Student stu0 = new Student();

// Create an object of class Person
//and call the public methods  to print the default value

Person p1=new Person("ahmad0",31,"tartus", false);// p1.printAllDatails();
//System.out.println("---");
p1.printAllDatails();
/* Create  an object of class Student and call the public method printS() to print
its data*/
Student stu1=new Student("ahmad",33.5,"lattakia", true, 4, "IT", 4.2);
stu1.printAllDatails();
```

جَامعة المَنارة

Protected Members

```
Employee e1= new Employee("ali",22.5, "hama", true, 200000,1.1,"Engineer");
e1.printAllDatails();
/* Appeal Override printAllDatails method 1 */
//Create an object of class SalaredEmployee and call the public methods

SalaredEmployee se1= new SalaredEmployee("adam",33.3, "syr",true , 3000,2.2,
"Eng", 700,200);
se1.printAllDatails();

// object of type Employer that also reference to an object of type Employee
Employee e11= new Employee("nader",22.5,"tartus", true, 300000,3.1,"Engineer");
e11.printAllDatails();

//object of type Employer that also reference to an object of type SataredEmployee
Employee e2= new SalaredEmployee("adam",33.3, "syr",true , 3000,2.2, "Eng",
700,200); e2.printAllDatails(); /*1 */
// getSalary() from SalaredEmployee Here, the method must have a root in the base
class, otherwise it will not be called
```

جامعة المنارة
MANARA UNIVERSITY

Protected Members

```
//A reference from derived class cannot refer to a base class

System.out.println("\n\n");
HourlyEmployee hE1= new HourlyEmployee("Adam2", 33, "latakia", true, 10000, 1.1,
"eng", 20,100);

hE1.printAllDatails();

}// end main
```

Protected Members

```
def. con. Parent run "super class"
-
name = null
Age 0.0
Address null
ationality = false

par. con. Parent run "super class"
-
name = Ahmad
Age 31.0
Address Syr. tartus
ationality = false

par. con. Parent run "super class"
par. con. Student run "sub class"
-
name = Hakim
Age 33.5
Address Syr. lattakia
ationality = true

stadyLevel = 4
specialization IT
GPA 4.2

par. con. Parent run "super class"
par. con. Employee run "sub class"
-
name = Ali
Age 22.5
Address Syr. hama
ationality = true

rank 1.1
job Engineer
salary = 200000.0

par. con. Parent run "super class"
par. con. Employee run "sub class"
par. con. SatEmployee run "sub class"

-
name = Hana
Age 33.3
Address Lib. beirut
ationality = true

rank 2.2
job Copywriter
salary = 3500.0
bonus = 700.0
deduction 200.0Salary = 3500.0

par. con. Parent run "super class"
par. con. Employee run "sub class"

-
name = Nader
Age 22.5
Address Syr. Damascus
ationality = true

rank 3.1
job Engineer
salary = 300000.0

par. con. Parent run "super class"
par. con. Employee run "sub class"
par. con. SatEmployee run "sub class"

-
name = Adam
Age 33.3
Address Jor. Amman
ationality = true
rank 2.2
job manager
salary = 3500.0
bonus = 700.0
deduction 200.0Salary = 3500.0

par. con. Parent run "super class"
par. con. Employee run "sub class"
par. con. HouEmployee run "sub class"

-
name = Mohamad
Age 33.0
Address Egy. Cairo
ationality = true

rank 1.1
job Technical
salary = 2000.0

houreRate = 20.0
numberOfHours 100.0
Salary = 2000.0
```

Overriding method (replacement or expansion)

- يوجد نوعين لإعادة تعريف الطرائق Overriding
  هما الاستيدال replacement والتوسيع expansion-

- الاستيدال replacement: تطرقنا لإعادة كتابة طريقة ()get_salary سابقاً، وقلنا إن الكائن من إي صنف هو الذي يحدد الطريقة التي سيناديها والموجوده بنفس الصنف، وفي حال الرغبة بمنادات الطريقة من صنف الأب تسبق ; ()super.get_salary واستخدمت طريقة الاستبدال نظراً لإن كل طريقة جديدة هي استبدال للطريقة الموروثة.

- يستخدام constructors طريقة التوسيع، (مثال أخر: استخدام طريقة ()printAllDatails لطباعة معلومات Person وتوسيعها ضمن الصنف Employee لإكمال مايخص الموظف هنا أول سطر سيكون منادادة طريقة طباعة Person وفق التالي ;()super. printAllDatails ثم كتابة مايخص الموظف)، عند اشتقاق كائن من الموظف ومناداة طريقة الطباعة سينفذ طريقة الأم ويكمل بتنفيذ طريقة الابن.

Reference type and object type

جَـامعة المَـنارة
MANARA UNIVERSITY

- ليكن لدينا مرجع من نوع موظف **e1** ويشير لكائن من نوع موظف.

Employee **e1** =new Employee (“adam”,30,”Hama”, …);

- ليكن لدينا مرجع من نوع Employee، **e2** ويشير لكائن من صنف SalariedEmployee (موظف شهري صنف مشتق من الأول).

حيث أن لكل منها المعادلة التي تحسب راتبه.

Employee **e2** =new SalariedEmployee (“mona”,20,”Aleppo”, …, , ,800,50);

- عند مناداة الطريقة ;()**e1**.get_salary سيتم تنفيذ الطريقة الموجوده ضمن Employee

وبالمناداة ;()**e2**.get_salary سيتم تنفيذ الطريقة ضمن SalariedEmployee أي أن الكائن هو من يحدد أية طريقه سيتم تنفيذها.

- إن ()get_salary موجوده ضمن الصنف الأب وبالتالي معرفة على كل الأصناف الوارثة له.
- يفرض أن طريقة معرفة ضمن SalariedEmployee وتم نداءها من **e2** لن يتم التعرف عليها رغم أن الكائن من نفس الصنف الموجوده به الطريقة، إن نوع المرجع **e2** هو Employee ولن يتعرف إلا على الطرق التي لها تعريف ضمن الصنف الأساس Employee.
- لايمكن لكائن من الصنف المشتق SalariedEmployee أن يؤشر إلى كائن من :

SalariedEmployee **e3**= new Employee (“adam”,30,”Hama”, …); // X

جامعة المنارة
MANARA UNIVERSITY

انتهت محاضرات الأسبوع 3