# INTERFACE DAN ABSTRACT

## Tujuan

Setelah mengikuti praktikum ini mahasiswa diharapkan dapat :

1. **Memahami** dan **mengerti** konsep **interface** dan mampu **mengimplementasikan** dalam Java.
2. **Memahami** dan **mengerti** konsep **abstract** dan mampu **mengimplementaskan** dalam Java.
3. Mampu **membedakan** antara class **abstract** dan **interface.**

## Interface

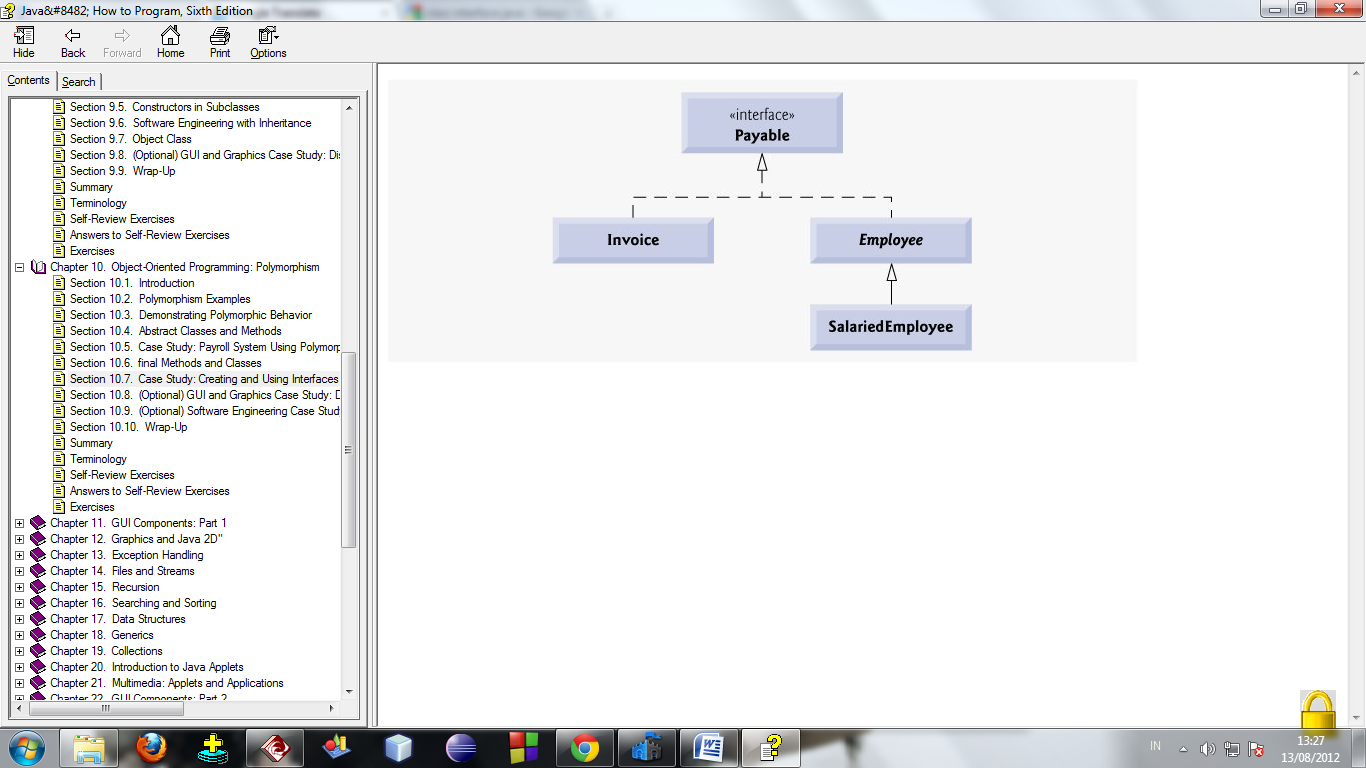
Interface serupa dengan class. **Class** mendefinisikan **kemampuan** sebuah **object**, sementara **interface** **mendefinisikan method** atau **konstanta** yang akan **diimplementasikan** pada **object** yang lain. Interface membantu **mendefinisikan** **sifat** object dengan **mendeklarasikan** seperangkat **karakteristik** object tersebut. Sebagai contoh kasus, radio, TV, dan speaker komputer memiliki pengontrol volume. Untuk alasan ini, mungkin kita menginginkan device-device ini mengimplementasikan interface yang bernama VolumeControl.

Interface memiliki beberapa batasan:

* **Semua** **atribut** adalah ***public, static*** dan ***final*** (semua atribut bertindak sebagai konstanta)
* **Semua** ***method*** adalah ***abstract*** dan ***public.***
* **Tidak** boleh **ada** **deklarasi konstruktor**.

Berbeda dengan class biasa, **interface** mengizinkan ***multiple inheritance****,* yaitu sebuah **class** dapat mempunyai **dua interface induk** sekaligus. Hal tersebut **tidak** dapat **dilakukan** pada sebuah **class,** dimana sebuah **class** **hanya** boleh mempunyai **satu class induk**.

**Syntax** yang diperlukan **hampir sama dengan** cara **membuat class**. **Perbedaannya** hanya **method** pada **interface** **tidak memiliki** **isi**/ **variable.** Sebuah contoh dengan tiga item: interface, class yang mengimplementasikan interface, dan class yang menggunakan class yang disalurkan. Sebagai ilustrasi perhatikan gambar UML berikut:



Gambar 5‑1 Contoh kelas interface dalam UML

Berdasarkan gambar diatas hirarki dimulai dengan interface Payable . **Relasi UML** yang terlihat antara **kelas** dan **interface** dinamakan ***realization***. Sebuah **kelas** dikatakan **merealisasikan** atau **mengimplementasikan** **method** pada sebuah **interface**.

Berikut contoh program dari UML diatas (Hanya interface Payable dan Kelas Invoice) .

Interface Payable

|  |
| --- |
| 4 public interface Payable  5 {  6 double getPaymentAmount(); // calculate payment; no implementation  7 } // end interface Payable |

Kelas Invoice yang mengimplementasikan Payable

|  |
| --- |
| public class Invoice implements Payable  5 {  6 private String partNumber;  7 private String partDescription;  8 private int quantity;  9 private double pricePerItem;  10  11 // four-argument constructor  12 public Invoice( String part, String description, int count,  13 double price )  14 {  15 partNumber = part;  16 partDescription = description;  17 setQuantity( count ); // validate and store quantity  18 setPricePerItem( price ); // validate and store price per item  19 } // end four-argument Invoice constructor  20  21 // set part number  22 public void setPartNumber( String part )  23 {  24 partNumber = part;  25 } // end method setPartNumber  26  27 // get part number  28 public String getPartNumber()  29 {  30 return partNumber;  31 } // end method getPartNumber  32  33 // set description  34 public void setPartDescription( String description )  35 {  36 partDescription = description;  37 } // end method setPartDescription  38  39 // get description  40 public String getPartDescription()  41 {  42 return partDescription;  43 } // end method getPartDescription  44  45 // set quantity  46 public void setQuantity( int count )  47 {  48 quantity = ( count <0 ) ? 0 : count; // quantity cannot be negative  49 } // end method setQuantity  50  51 // get quantity  52 public int getQuantity()  53 {  54 return quantity;  55 } // end method getQuantity  56  57 // set price per item  58 public void setPricePerItem( double price )  59 {  60 pricePerItem = ( price <0.0 ) ? 0.0 : price; // validate price  61 } // end method setPricePerItem  62  63 // get price per item  64 public double getPricePerItem()  65 {  66 return pricePerItem;  67 } // end method getPricePerItem  68  69 // return String representation of Invoice object  70 public String toString()  71 {  72 return String.format( "%s: \n%s: %s (%s) \n%s: %d \n%s: $%,.2f",  73 "invoice", "part number", getPartNumber(), getPartDescription(),  74 "quantity", getQuantity(), "price per item", getPricePerItem() );  75 } // end method toString  76  77 // method required to carry out contract with interface Payable  78 public double getPaymentAmount()  79 {  80 return getQuantity() \* getPricePerItem(); // calculate total cost  81 } // end method getPaymentAmount  82 } // end class Invoice |

public interface Product {

static final String MAKER = "My Corp";

static final String PHONE = "555-123-4567";

public int getPrice(int id);

}

public class Shoe implements Product {

public int getPrice(int id) {

if (id == 1)

return(5);

else

return(10);

}

public String getMaker() {

return(MAKER);

}

}

public class Store {

static Shoe hightop;

public static void init() {

hightop = new Shoe();

}

}

public static void main(String argv[]) {

init();

getInfo(hightop);

orderInfo(hightop);

}

public static void getInfo(Shoe item) {

System.out.println("This Product is made by "+ item.MAKER);

System.out.println("It costs $" + item.getPrice(1) + '\n');

}

public static void orderInfo(Product item) {

System.out.println("To order from "+item.MAKER+" call "+item.PHONE+".");

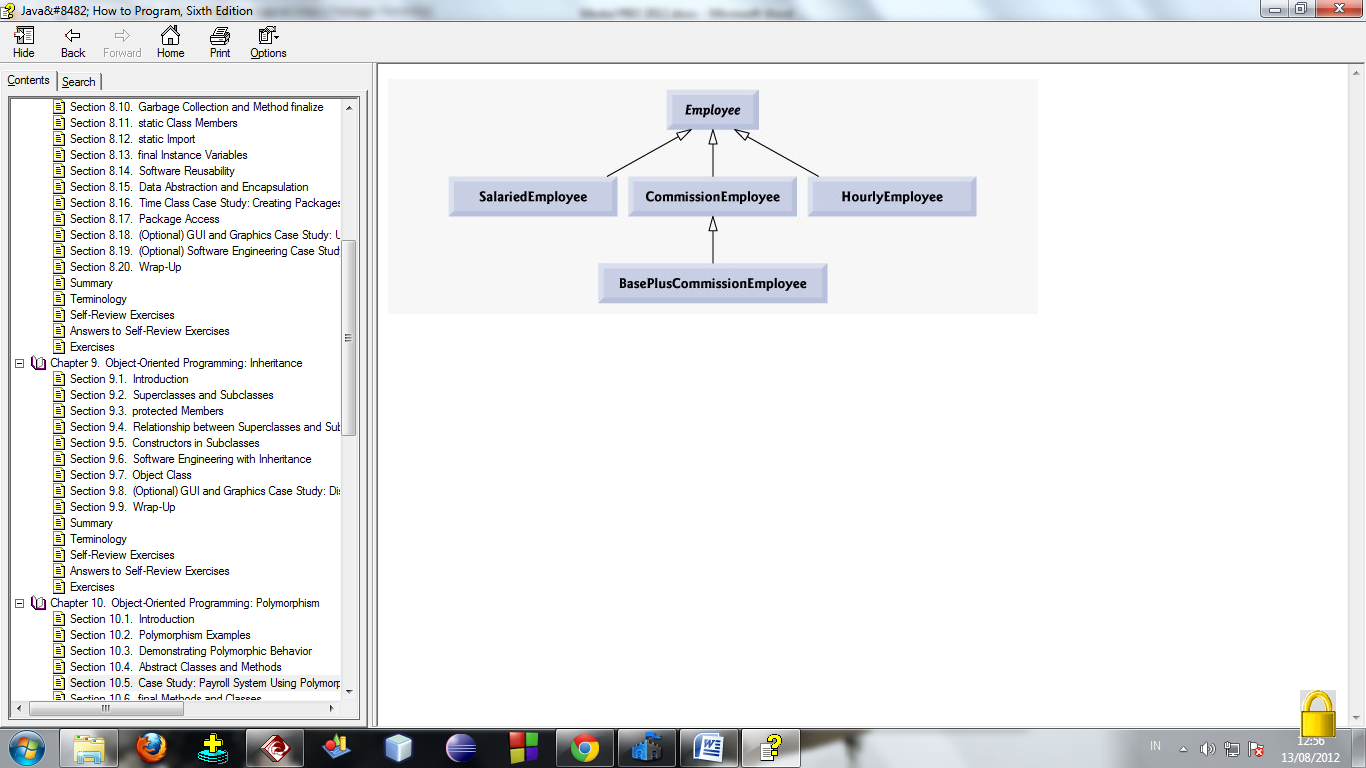
System.out.println("Each item costs $"+item.getPrice(1));

}

}

## Kelas Abstrak

Kata kunci **abstrak** digunakan untuk **method** atau **class** yang **belum** memiliki **implementasi**. **Abstrak** **method** **dideklarasikan** pada **abstrak class**. **Class** yang dideklarasikan sebagai **abstrak** **tidak** akan bisa **dibentuk object** dalam Java. Jadi **class abstrak** **tidak** akan bisa **digunakan** di dalam **program** kita yang **lain**, **kecuali** class **abstrak** tersebut **diturunkan** dan **turunan** dari class **abstrak** tersebut yang **bisa** **dijadikan object**. Sebagai ilustrasi dari kelas abstrak perhatikan gambar di bawah ini :



Gambar 5‑2 Contoh kelas abstrak dalam UML

G**ambar 5-2** **menggunakan** kelas **Employee** untuk **merepresentasikan** **konsep** umum **pegawai**. Kelas **Employee** **diperluas** ke kelas **SalariedEmployee, CommissionEmployee** dan **HourlyEmployee**. **BasePlusCommissionEmployee** yang **memperluas CommisionEmployee** merepresentasikan **tipe** akhir **pegawai.** Berdasarkan gambar kelas **Employee** **diketik miring** **menandakan** bahwa **kelas** tersebut merupakan **kelas abstrak.**

Berikut contoh program dari kelas abrak employee dan Kelas SalariedEmployee.

Kelas Abstrak Employee

|  |
| --- |
| public abstract class Employee  5 {  6 private String firstName;  7 private String lastName;  8 private String socialSecurityNumber;  9  10 // three-argument constructor  11 public Employee( String first, String last, String ssn )  12 {  13 firstName = first;  14 lastName = last;  15 socialSecurityNumber = ssn;  16 } // end three-argument Employee constructor  17  18 // set first name  19 public void setFirstName( String first )  20 {  21 firstName = first;  22 } // end method setFirstName  23  24 // return first name  25 public String getFirstName()  26 {  27 return firstName;  28 } // end method getFirstName  29  30 // set last name  31 public void setLastName( String last )  32 {  33 lastName = last;  34 } // end method setLastName  35  36 // return last name  37 public String getLastName()  38 {  39 return lastName;  40 } // end method getLastName  41  42 // set social security number  43 public void setSocialSecurityNumber( String ssn )  44 {  45 socialSecurityNumber = ssn; // should validate  46 } // end method setSocialSecurityNumber  47  48 // return social security number  49 public String getSocialSecurityNumber()  50 {  51 return socialSecurityNumber;  52 } // end method getSocialSecurityNumber  53  54 // return String representation of Employee object  55 public String toString()  56 {  57 return String.format( "%s %s\nsocial security number: %s",  58 getFirstName(), getLastName(), getSocialSecurityNumber() );  59 } // end method toString  60  61 // abstract method overridden by subclasses  62 public abstract double earnings(); // no implementation here  63 } // end abstract class Employee |

Kelas SalariedEmployee yang merupakan turunan dari Kelas Abstrak.

|  |
| --- |
| public class SalariedEmployee extends Employee  5 {  6 private double weeklySalary;  7  8 // four-argument constructor  9 public SalariedEmployee( String first, String last, String ssn,  10 double salary )  11 {  12 super( first, last, ssn ); // pass to Employee constructor  13 setWeeklySalary( salary ); // validate and store salary  14 } // end four-argument SalariedEmployee constructor  15  16 // set salary  17 public void setWeeklySalary( double salary )  18 {  19 weeklySalary = salary <0.0 ? 0.0 : salary;  20 } // end method setWeeklySalary  21  22 // return salary  23 public double getWeeklySalary()  24 {  25 return weeklySalary;  26 } // end method getWeeklySalary  27  28 // calculate earnings; override abstract method earnings in Employee  29 public double earnings()  30 {  31 return getWeeklySalary();  32 } // end method earnings  33  34 // return String representation of SalariedEmployee object  35 public String toString()  36 {  37 return String.format( "salaried employee: %s\n%s: $%,.2f",  38 super.toString(), "weekly salary", getWeeklySalary() );  39 } // end method toString  40 } // end class SalariedEmployee |

Perbedaan ***abstract class*** dan ***interface*** antara lain:

1. Semua ***interface method* tidak memiliki *body*** sedangkanbeberapa ***abstract class* dapat memiliki *method*** dengan **implementasi.**
2. Sebuah ***interface* hanya** dapat **mendefinisikan *constant*** sedangkansebuah ***abstract class* tampak** seperti **class biasa** yang **dapat mendeklarasikan variable.**
3. ***Interface* tidak memiliki hubungan *inheritance* secara langsung dengan sebuah *class*** tertentu**,** sedangkan ***abstract class* bisa jadi hasil turunan dari *abstract class* induknya.**
4. ***Interface* memungkinkan terjadinya pewarisan jamak (*Multiple inheritance*) sedangkan *abstract class* tidak.**

## Keyword super

Keyword **super digunakan** untuk **merefer superclass** dari suatu **class**, yaitu untuk **merefer member** dari suatu **superclass**, baik **atribut** maupun **method**.

Contoh :



Gambar 5‑3 Contoh kelas dengan keyword super

Dari UML diatas dapat dilihat bahwa kelas **Employee** merupakan **superclass** dari kelas **Manager**.

|  |
| --- |
| class Employee {  private String name;  String departemen;  public Employee (String s) {  name = s;  }  public void tampilNama()  {  System.out.println("nama : "+name);  }  }  class Manager extends Employee {  private String alamat;  public Manager(String nama, String s) {  /\*memanggil konstruktor employee\*/  super(nama);  alamat = s;  }  public void tampilAlamat()  {  /\*menginisialisasi variabel departemen yang ada pada superclass\*/  super.departemen="Personalia";  /\*memanggil method tampilNama() yang ada pada superclass\*/  super.tampilNama();  /\*menampilkan variabel departemen yang telah diinisialisasi\*/  System.out.println("alamat : "+alamat);  System.out.println("departemen : "+super.departemen);  }  public static void main(String[] args)  {  /\*membuat objek\*/  Manager gufron = new Manager("gufron","buahbatu"); gufron.tampilAlamat();  }  } |

Output Program :

|  |
| --- |
| Nama : gufron  Alamat : buahbatu  Departemen : Personalia |

***“Abstract class itu digunakan untuk mengimplementasikan pattern Template Method. Sedangkan interface digunakan (diantaranya) untuk mengimplementasikan pattern Observer”***