**interface**

* Interface adalah kumpulan method-method tanpa implementasi bahkan tidak boleh mempunyai body
* Tujuan dibuat interface adalah untuk memaksa kelas-kelas yang mengimplementasikan interface tersebut melakukan overide method abstrak milik interface
* Interface adalah semacam kelas abstract tetapi hanya boleh mempunyai anggota method abstract dan variabel konstan
* Interface biasa digunakan untuk standarasiasi method atau konstanta
* Jika kita tidak mengimplementasikan semua method yang ada pada interface, maka class tersebut harus dideklarasikan sebagai abstract class

**class Testing** {

  public static void main (String [] args)

 {

  Berita news = new Berita();

  News.cetak();

 }

}

**interface HargaBBM** {

 int harga = 4500;

  public void cetak();

}

**class Berita implements hargaBBM()** {

  private int hargaNOW = 5000;

  public void cetak() {

  if (this.hargaNOW == harga) {

  System.out.println(“Harga tetap”);

 }

 else {

  System.out.println(“Harga naik !!”);

 }

 }

}

**Kelas abstract**

* Kelas abstract hanya berisi deklarasi method tapi tanpa mendefinisikan. Definisi method didefinisikan di sub classnya dan class abstract tidak bisa digunakan untuk membuat objek.
* class abstract mendefinisikan method-method atau variabel yang akan dibutuhkan sub-sub classnya.
* Kelas abstract adalah kelas yang memakai modifier abstract
* Di dalam kelas abstract bisa berisi variabel anggota maupun method abstract maupun kongkrit
* Kelas abstract tidak bisa diinstansiasi, hanya bisa diextends

// KotaWisata.java

**public abstract class KotaWisata** {

  String objekWisata;

  public abstract Jalan();

}

//Liburan.java

**class Jogja extends KotaWisata** {

  public Jogja (String objekWisata) {

 this. objekWisata = objekWisata;

 }

 public void Jalan() {

  System.out.println(“Jogja objek wisatanya candi ” + this.objekWisata)

 }

}

**class Semarang extends KotaWisata** {

  public Semarang (String objekWisata) {

  this. objekWisata = objekWisata;

 }

 public void Jalan() {

  System.out.println(“Semarang objek wisatanya ” + this.objekWisata)

 }

}

**class Liburan {**

  public static void main(String [] args) {

  Jogja jgj = new Jogja(“Prambanan”);

  jgj.Jalan();

  Semarang Srg = new Semarang(“Tugu Muda”);

  Srg.Jalan();

 }

}

***Abstract class*** adalah sebuah class setengah jadi (abstrak) yang memuat/memiliki method dan atrribut.
*Abstract class* sebenarnya adalah sebuah class, sehingga memiliki semua sifat dari class biasa (punya konstruktor). Hanya saja sifatnya masih abstrak, karena itu biasanya method kosong/belum di implementasikan.
Namun *Abstract class* dapat mengimplementasikan method tersebut.
*Abstract class* akan selalu menjadi superclass / hirarki tertinggi dari subclass-subclass-nya.

**Sedangkan*interface:***
- Adalah sebuah blok signature kumpulan method tanpa tubuh (konstan).
- Sebuah definisi method yang umum/*general* yang dapat menghubungkan class-class yang berbeda.
Dengan kata lain, *interface*memungkinkan kita mengimplementasikan method yang sama terhadap class yang tidak ada hubungan sama sekali (tidak dalam satu hirarki)
- Oleh karena itu *interface*bukanlah sebuah class, walaupun memiliki ciri yang serupa dengan *abstract class*.

Interface merupakan suatu mekanisme yang disediakan oleh java yang

memungkinkan berbagi konstanta atau menentukan bentuk metode yang dapat digunakan

oleh sejumlah kelas.

Beda Interface dan kelas abstrak:

Kelas abstrak dapat mengandung metode abstrak maupun metode tidak abstrak, sedangkan interface hanya dapat mengandung metode abstrak.

Kelas abstrak dapat mendeklarasikan variabel instan, sedangkan interface hanya dapat mendefinisikan konstanta.

Kelas abstrak digunakan oleh kelas lain melalui pewarisan(via kata kunci extends), sedangkan interface digunakan oleh kelas tidak melalui mekanisme itu.

**ABSTRACT CLASS**

Kelas abstrak adalah kelas yang terletak pada posisi tertinggi hierarki kelas.

Kelas ini digunakan sebagai basis bagi penurunan kelas-kelas lainnnya, sehingga tidak dapat diinstansiasi secara langsung menjadi objek. Sebagai kelas basis kelas ini hanya berisi variable-variable umum dan deklarasi method tanpa detail implementasi. Selanjutnya, kelaskelas yang diturunkan dari kelas abstraklah yang harus mendefinisikan secara detail method tersebut.

Kelas abstrak adalah kelas yang mengandung konsep abstrak sehingga tidak mungkin mempunyai instans. Misalnya suatu kelas abstrak Buah yang mengandung konsep tentang bagian dari tumbuhan yang dapat dimakan. Namun kita tidak dapat menciptakan sebuah instans dari kelas tersebut karena tidak masuk akal menciptakan suatu Buah. Yang mungkin adalah menciptakan instans dari kelas Jeruk, Apel, atau kelas lain yang sudah mengimplementasikan konsep abstrak dari buah. Kelas abstrak dapat mengandung metode abstrak, yaitu metode yang tidak memiliki implementasi. Dengan begitu, kelas abstrak dapat menentukan bagaimana konsep

abstrak tersebut diimplementasikan oleh subkelas yang akan menggunakannya. Kelas abstrak tidak harus memiliki metode abstrak, namun setiap kelas yang memiliki metode abstrak haruslah menjadi kelas abstrak.

Jadi :

Abstract class adalah class yang mempunyai setidaknya satu abstract method.

Abstract method adalah method yang tidak memiliki body ( hanya deklarasi method ).

Implementasi dari isi abstract method tersebut dilakukan pada subclass

Subclass harus mengimplementasikan semua method abstrak.

Bila subclass method abstract tidak mengimplementasikan isi semua method maka subclass tersebut harus dideklarasikan abstract. Dan deklarasi method abstract pada subclass tersebut boleh tidak dituliskan kembali.

Abstract class tidak bisa dibuat objectnya

Object hanya bisa dibuat dari non-abstract class (concrete class)

Konsekuensinya, suatu abstract class haruslah diturunkan dimana pada subclass tersebut berisi implementasi dari abstract method yang ada di superclass.

|  |  |
| --- | --- |
| **Java source code** | **UML visual representation** |
| The image shows the sample Java source code. | The image shows how an implementation relationship is represented in UML visualization diagrams. |