1. Jelaskan perbedaan kelas abstrak dan interface!

Jawaban :

//abstrak

//1. Merupakan hirarki paling teratas dari kelas induk

//2. Boleh mempunyai kelas anak

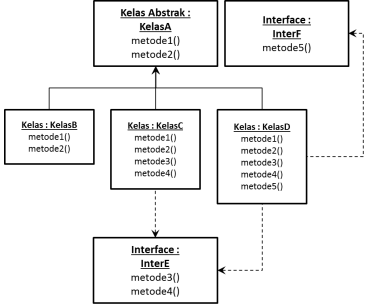
//3. Kelas abstrak tidak boleh diinstansiasi

//interface

//1. kelas yang tidak ada hubungannya dengan konsep pewarisan

//2. mengikat aturan kepada kelas-kelas, tanpa terhubung dengan konsep pewarisan

2. Sebutkan dan jelaskan kegunaan dan konsep yang digunakan pada masing-masing kelas berdasarkan gambar dibawah ini :

[](https://www.blogger.com/u/1/blog/post/edit/4310209025248791298/228145150090591893)

Jawaban :

//KelasA merupakan kelas abstrak yang terhubung dengan kelas induk

//kelas induk ada KelasB, KelasC, KelasD

//kelasB akan mendapat warisan metode dari kelas abstrak

//KelasC juga sama, akan tetapi mempunyai metode tambahan

//KelasC metodenya yang sama dengan kelas InterE akan di update oleh kelas interface

//KelasD sama seperti kelas C akan tetapi metode lebih banyak dari metode KelasC

//KelasD metodenya yang sama dengan kelas InterF akan di update oleh kelas interface

3. Buatlah program berdasarkan gambar soal B, dengan ketentuan sebagai berikut :

1. Semua metode yang kita gunakan bertipe void.

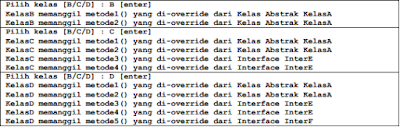
2. Semua metode tidak memiliki argumen.

3. Isi metode pada masing-masing kelas memiliki tampilan teks yang berbeda-beda. Berikut formatnya :

[](https://www.blogger.com/u/1/blog/post/edit/4310209025248791298/228145150090591893)

4. Buat sebuah kelas utama yang akan meinstansiasi objek dari kelas KelasB, KelasC dan KelasD.

Contoh Output:

[](https://www.blogger.com/u/1/blog/post/edit/4310209025248791298/228145150090591893)

Jawaban :

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

abstract class KelasA{

     String KlsA="KelasA";

     public void method1(String metode1){

          String metode1="metode1";

     }

     public void method2(String metode2){

          String metode2="metode2";

     }

}

class KelasB extends KelasA{

     String KlsA="KelasA";

     String KlsB="KelasB";

     public void method1(String metode1){

          super.metode1 = metode1;

     }

     public void method2(String metode2){

          super.metode2 = metode2;

     }

     void show(){

          System.out.println(this.KlsB+" memanggil "+this.metode1+" yang di-override dari <Kelas Abstrak/Interface>");

          System.out.println("KelasB "+this.metode2);

     }

}

class KelasC extends KelasA{

     void method3();

     void method4();

     void show(){

          System.out.println("KelasC "+super.metode1);

          System.out.println("KelasC "+super.metode2);

          System.out.println("KelasC "+InterE.metode3);

     }

}

class KelasD extends KelasC{

}

public interface InterE implements KelasC,KelasD{

     void method3(){

          System.out.println("KelasC memanggil metode3() yang di-override dari Interface InterE");

     }

     void method4(){

          System.out.println("KelasC memanggil metode4() yang di-override dari Interface InterE");

     }

}

public interface InterF{

}

class Program{

     public static void main(String[] args){

          char pilih;

          try{

              BufferedReader input = new

              BufferedReader(new InputStreamReader(System.in));

              System.out.println("Pilih kelas [B/C/D] :");

              pilih = input.readLine().charAt(0);

              if(pilih=="B"||pilih=="b"){

                   KelasB x = new KelasB();

                   x.show();

              }

              else if(pilih=="C"||pilih"c"){

                   KelasB x = new KelasB();

                   KelasC y = new KelasC();

              }

              else if(pilih=="D"||pilih=="d"){

                   KelasB x = new KelasB();

                   KelasC y = new KelasC();

                   KelasD z = new KelasD();

              }

              else{

                   System.out.print("Aplikasi keluar");

              }

          }

     }

}

/\*

abstract class KelasA{

     public abstract void method1();

     public abstract void method2();

}

class KelasB extends KelasA{

     String KlsA="Kelas Abstrak KelasA";   String Kls="KelasB";

     String metode1="metode1",metode2="metode2";

     public void method1(){

          System.out.println(Kls+" memanggil "+metode1+" yang di-override dari "+KlsA);

     }

     public void method2(){

          System.out.println(Kls+" memanggil "+metode2+" yang di-override dari "+KlsA);

          System.out.println();

     }

}

class KelasC extends KelasA{

     String KlsA="Kelas Abstrak KelasA";   String Kls="KelasC";

     String metode1="metode1",metode2="metode2";

     String metode3="metode3",metode4="metode4";

     String InterF="Interface InterE";

     public void method1(){

          System.out.println(Kls+" memanggil "+metode1+" yang di-override dari "+KlsA);

     }

     public void method2(){

          System.out.println(Kls+" memanggil "+metode2+" yang di-override dari "+KlsA);

          System.out.println();

     }

     void method3();

     void method4();

}

public interface intf{

     public void method3();

     public void method4();

}

public interF implements intf{

     String KlsA="null"; String Kls="null";

     String metode3="metode3",metode4="metode4";

     public String Kls(String Kls)

     {

        this.Kls = Kls;

    }

     public String KlsA(String KlsA)

     {

        this.KlsA = KlsA;

    }

     public void method3(){

          System.out.println(Kls+" memanggil "+metode3+" yang di-override dari "+KlsA);

     }

     public void method4(){

          System.out.println(Kls+" memanggil "+metode3+" yang di-override dari "+KlsA);

     }

}

class Program{

     public static void main(String[] args){

          KelasB x = new KelasB();

          x.method1();

          x.method2();

          KelasC y = new KelasC();

          y.method1();

          y.method2();

          interF p = new interF();

          p.Kls("KelasC"); p.Kls("Kelas interF");

          p.method3();

          p.method4();

     }

}

\*/