



Modul 09

Abstract Class dan Interface

I. Kompetensi

Setelah menyelesaikan modul ini mahasiswa diharapkan mampu:

1. Menjelaskan maksud dan tujuan penggunaan Abstract Class;
2. Menjelaskan maksud dan tujuan penggunaan Interface;
3. Menerapkan Abstract Class dan Interface di dalam pembuatan program.

II. Pendahuluan

Abstract Class

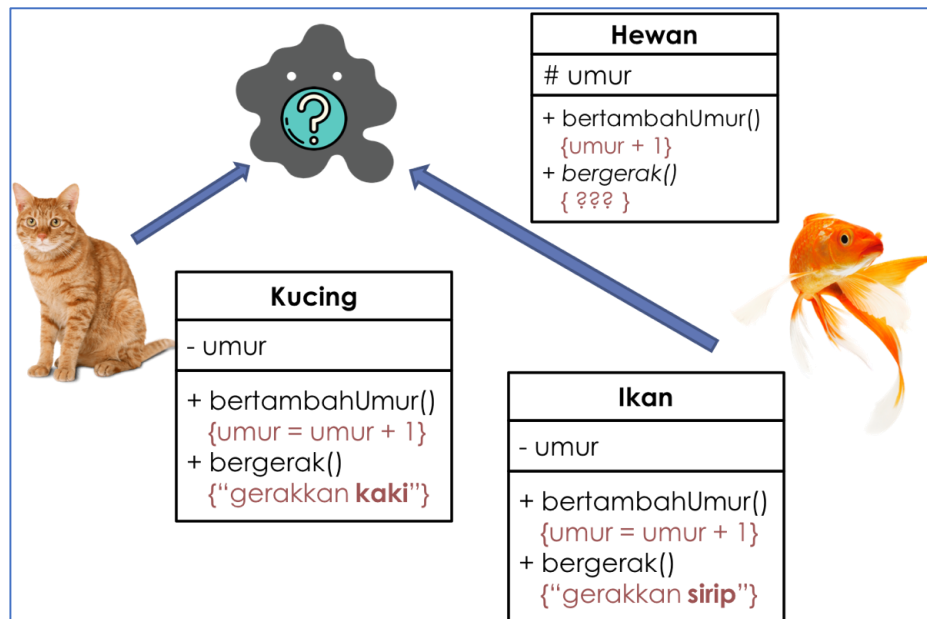
Abstract Class adalah class yang tidak dapat diinstansiasi namun dapat di-*extend*. *Abstract class baru* dapat dimanfaatkan ketika ia di-*extend*.

Karakteristik:

- a. Dapat memiliki *properties* dan *methods* seperti class biasa.
- b. Selalu memiliki *methods* yang tidak memiliki tubuh (hanya deklarasinya saja), disebut juga *abstract method*.
- c. Selalu dideklarasikan dengan menggunakan kata kunci `abstract class`.

Kegunaan:

Menggambarakan sesuatu yang bersifat umum, yang hanya bisa berfungsi setelah ia dideskripsikan ke dalam bentuk yang lebih spesifik.



Interface

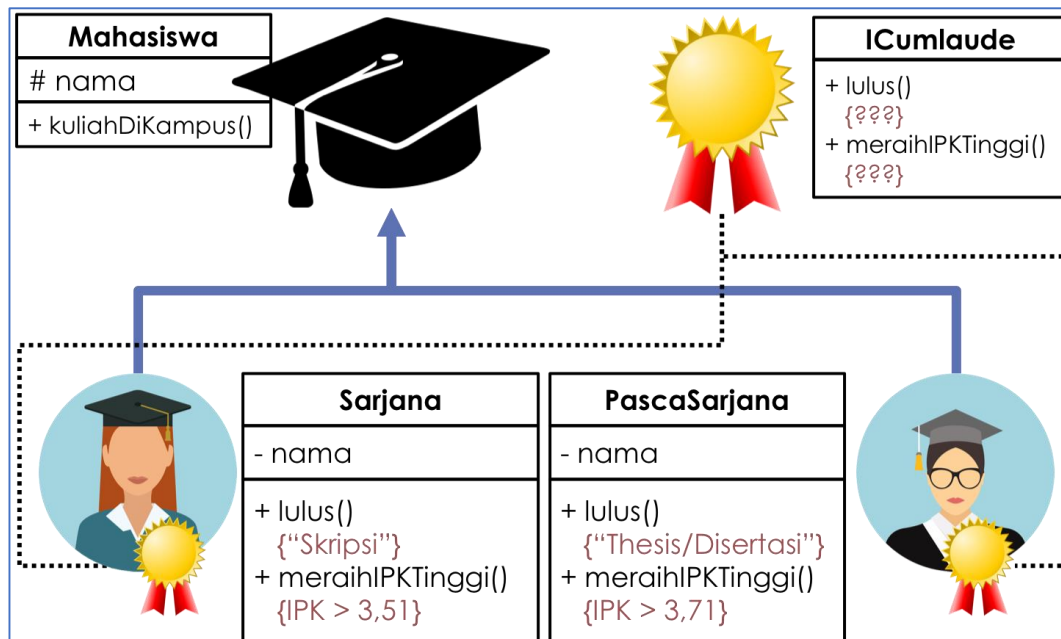
Interface adalah struktur data yang hanya berisi *abstract methods*. Tidak ada apa-apa selain *method abstract* pada *interface*, termasuk atribut *getter* dan *setter*.

Karakteristik:

- Tidak ada apa-apa di dalamnya selain *abstract methods*.
- Di konvensi bahasa pemrograman Java, namanya dianjurkan untuk selalu diawali dengan huruf kapital 'I'.
- Selalu dideklarasikan dengan menggunakan kata kunci `interface`.
- Diimplementasikan dengan menggunakan kata kunci `implements`.

Kegunaan:

Bertindak seperti semacam kontrak/syarat yang HARUS dipenuhi bagi suatu class agar class tersebut dapat dianggap sebagai 'sesuatu yang lain'.



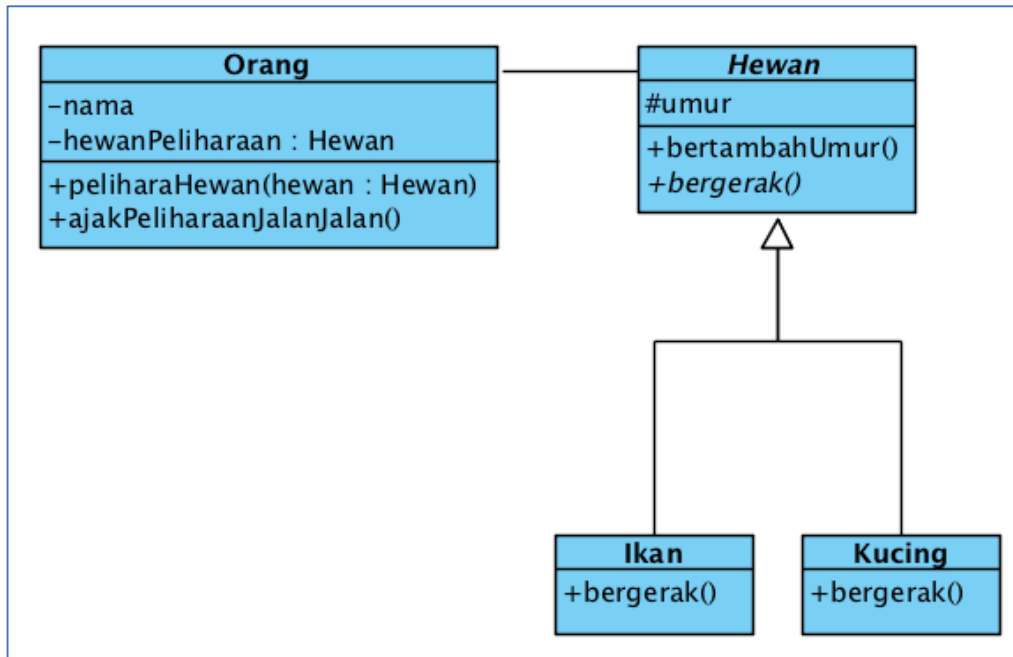
III. Praktikum

Percobaan 1: Abstract Class

Di dunia ini terdapat banyak jenis hewan. Semua hewan memiliki beberapa karakteristik yang sama, seperti contohnya semua hewan memiliki umur, hewan apapun itu, umurnya akan bertambah sama jumlahnya setiap tahun.

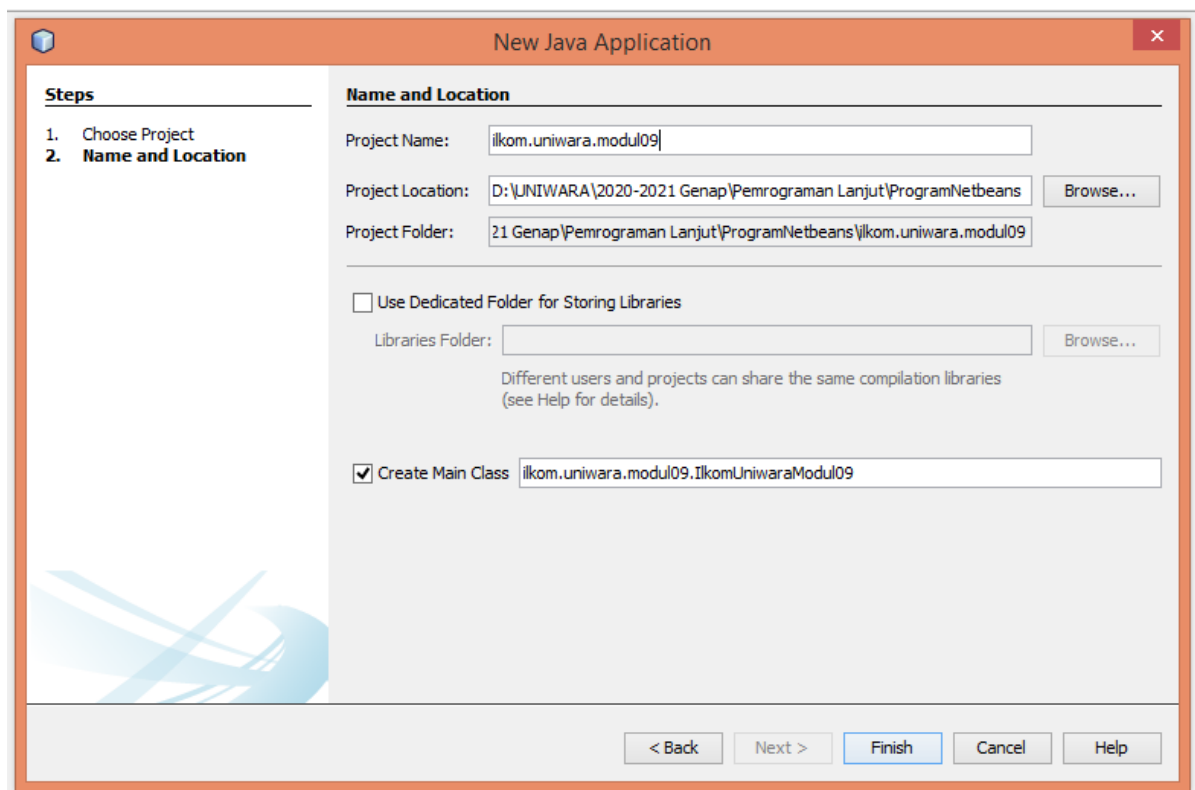
Selain karakteristik yang sama, masing-masing hewan juga memiliki karakteristik yang berbeda satu dengan yang lainnya. Contohnya dalam hal **bergerak**. Cara kucing bergerak berbeda dengan cara ikan bergerak. Kucing bergerak dengan cara melangkahakan kaki-kakinya sedangkan ikan bergerak dengan cara menggerakkan siripnya.

Setiap orang yang memelihara hewan dapat mengajak hewan peliharaannya berjalan (membuat agar hewan peliharaannya bergerak). Namun orang yang memelihara hewan yang berbeda, akan berbeda pula cara hewan peliharaannya dalam bergerak.

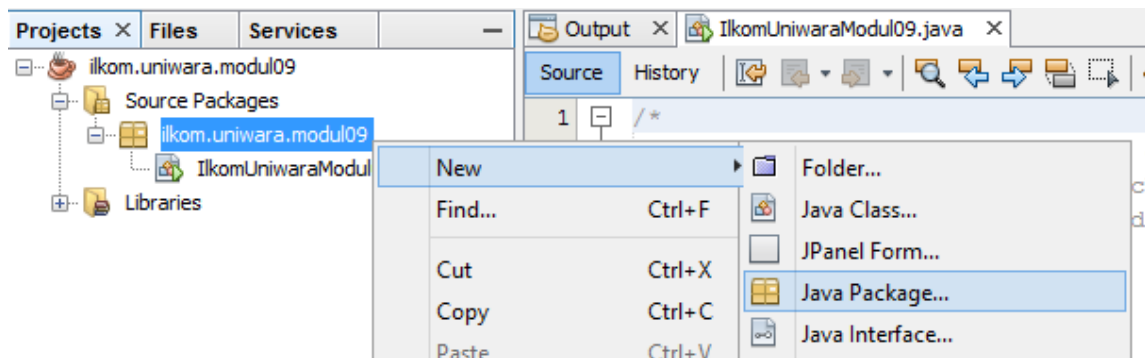


Pada percobaan pertama ini kita akan membuat sebuah program yang menggambarkan skenario di atas dengan memanfaatkan **abstract class**.

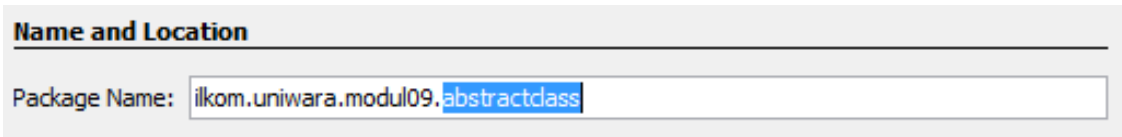
1. Buatlah sebuah project baru di NetBeans dengan nama **ilkom.uniwara.modul09**



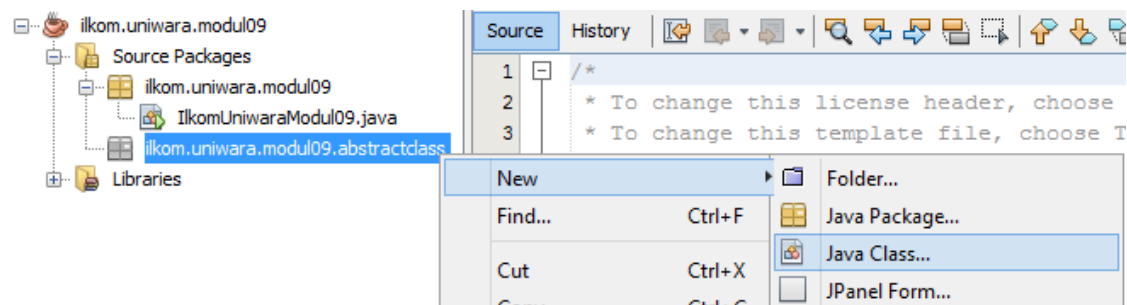
2. Pada package **ilkom.uniware.modul09**, tambahkan package baru dengan cara klik kanan nama package → New → Java Package...



3. Beri nama package tersebut dengan nama **abstractclass**. Semua class yang dibuat pada percobaan 1 ini **diletakkan pada package yang sama**, yaitu *package abstractclass* ini,



4. Pada package baru tersebut tambahkan *class* baru.



5. Beri nama *class* baru tersebut, yaitu class **Hewan**.





6. Pada *class* Hewan tersebut, ketikkan kode berikut ini.

```
public abstract class Hewan
{
    private int umur;

    protected Hewan()
    {
        this.umur = 0;
    }

    public void bertambahUmur()
    {
        this.umur += 1;
    }

    public abstract void bergerak();
}
```

Class Hewan tersebut adalah class abstract berisi property dan method biasa, ditambah sebuah *method abstract* bernama **bergerak()**. *Method* tersebut didepannya terdapat kata kunci **abstract** dan tidak memiliki badan fungsi. Method ini nantinya akan di-*override* oleh *class* mana saja yang menjadi *class* turunan dari class Hewan tersebut.

7. Dengan cara yang sama, buatlah class dengan nama **Kucing** yang meng-*extend* class Hewan. Di dalam class Kucing tersebut, setelah Anda menuliskan kode seperti di bawah, maka akan muncul ikon lampu peringatan. Klik lampu tersebut dan kemudian pilih **implement all abstract methods**.

```
13 public class Kucing extends Hewan{
14 }
```

8. Maka akan secara otomatis dibuatkan fungsi yang meng-*override* fungsi *abstract* **bergerak()** yang ada pada class hewan.

```
public class Kucing extends Hewan{

    @Override
    public void bergerak() {
        throw new UnsupportedOperationException("Not supported yet.");
    }

}
```



9. Ubahlah badan fungsi tersebut dengan mengganti kode didalamnya menjadi seperti berikut.

```
@Override
public void bergerak()
{
    System.out.println("Berjalan dengan KAKI, \"Tap..tap..\"");
}
```

10. Dengan cara yang sama seperti ketika Anda membuat class Kucing, buatlah class Hewan baru bernama **Ikan** dan buatlah kodenya seperti pada gambar dibawah.

```
public class Ikan extends Hewan{

    @Override
    public void bergerak() {
        System.out.println("Berenang dengan SIRIP, \"wush..wush..\"");
    }

}
```

11. Selanjutnya, buatlah class biasa baru yang bernama class **Orang**. Class ini adalah class yang menjadi pengguna dari *class abstract* Hewan yang sudah dibuat sebelumnya. Ketikkan pada class Orang tersebut, baris-baris kode seperti di bawah.

```
public class Orang
{
    private String nama;
    private Hewan hewanPeliharaan;

    public Orang(String nama)
    {
        this.nama = nama;
    }

    public void peliharaHewan(Hewan hewanPeliharaan)
    {
        this.hewanPeliharaan = hewanPeliharaan;
    }

    public void ajakPeliharaanJalanJalan()
    {
        System.out.println("Namaku " + this.nama);
        System.out.println("Hewan peliharaanku berjalan dengan cara: ");
        this.hewanPeliharaan.bergerak();
        System.out.println("-----");
    }
}
```



12. Terakhir, buatlah sebuah *Main Class* baru di dalam *package* yang sama. Beri nama class baru tersebut dengan nama class **Program**. Ketikkan didalamnya seperti kode di bawah ini.

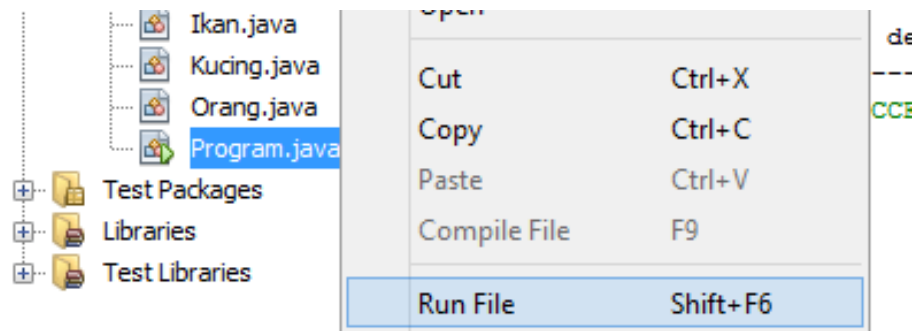
```
public class Program
{
    public static void main(String[] args)
    {
        Kucing kucingKampung = new Kucing();
        Ikan lumbaLumba = new Ikan();

        Orang ani = new Orang("Ani");
        Orang budi = new Orang("Budi");

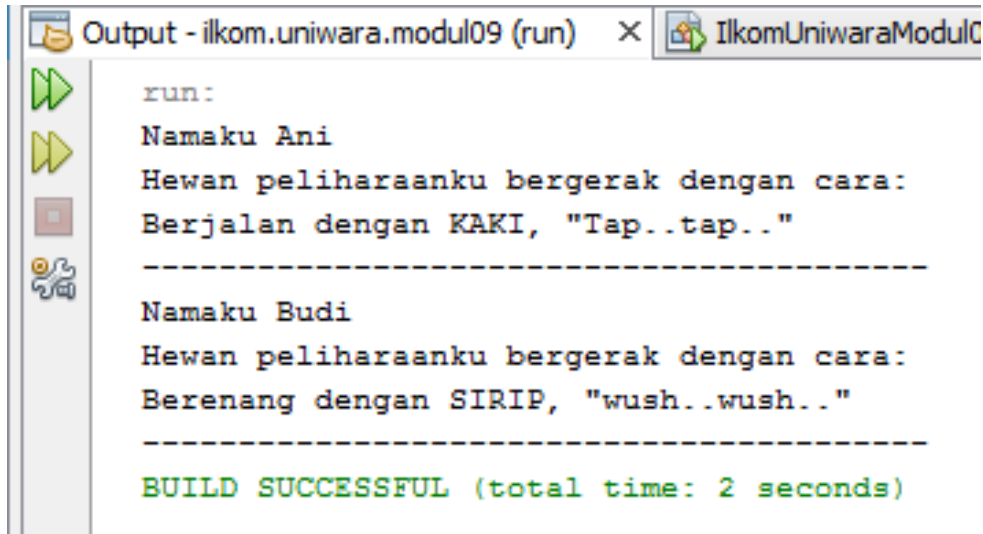
        ani.peliharaHewan(kucingKampung);
        budi.peliharaHewan(lumbaLumba);

        ani.ajakPeliharaanJalanJalan();
        budi.ajakPeliharaanJalanJalan();
    }
}
```

13. Jalankan class tersebut dengan cara klik kanan pada class Program kemudian pilih **Run File** (Shift + F6).



14. Perhatikan dan amati hasilnya!



```

run:
Namaku Ani
Hewan peliharaanku bergerak dengan cara:
Berjalan dengan KAKI, "Tap..tap.."
-----
Namaku Budi
Hewan peliharaanku bergerak dengan cara:
Berenang dengan SIRIP, "wush..wush.."
-----
BUILD SUCCESSFUL (total time: 2 seconds)

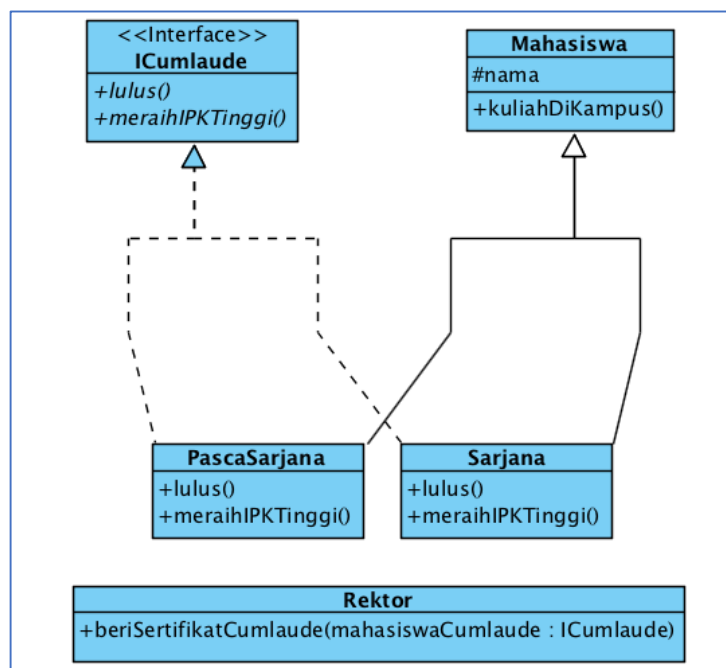
```

15. Pertanyaan diskusi:

Bolehkah apabila sebuah class yang meng-*extend* suatu *abstract class* tidak mengimplementasikan *method abstract* yang ada di class induknya? Buktikan!

Percobaan 2: Interface

Pada sebuah wisuda, seorang Rektor akan memberikan penghargaan sertifikat *Cumlaude* pada semua mahasiswa yang memenuhi persyaratan. Persyaratan agar seorang mahasiswa dapat disebut sebagai *Cumlaude* berbeda-beda antara mahasiswa Sarjana dan Pasca Sarjana.





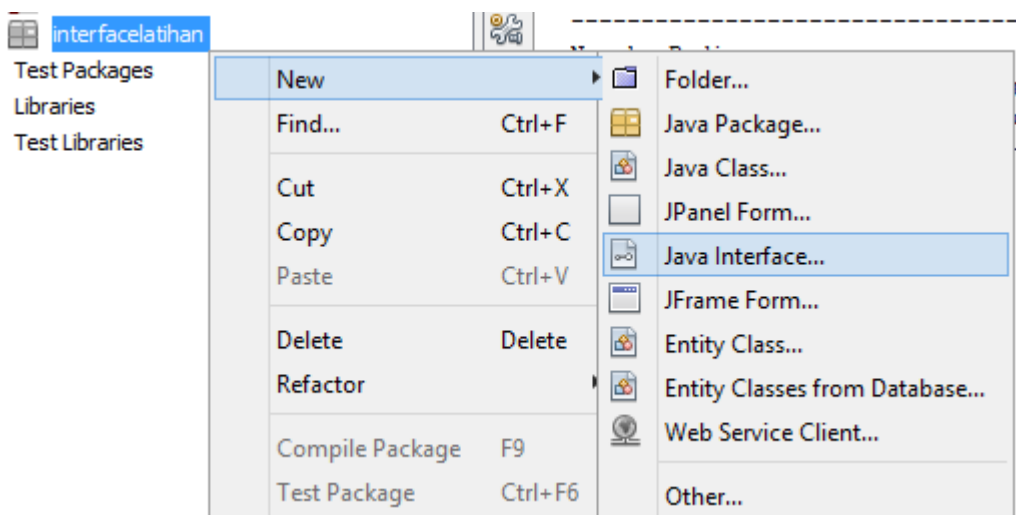
Untuk menjadi *cumlaude*, mahasiswa Sarjana harus mengerjakan **skripsi** dan memiliki IPK lebih tinggi dari 3,51. Sedangkan untuk mahasiswa Pasca Sarjana, mereka harus mengerjakan **tesis** dan meraih IPK lebih tinggi dari 3,71.

Pada percobaan ini kita akan mencoba menerjemahkan skenario di atas ke dalam sebuah aplikasi sederhana yang memanfaatkan interface.

1. Pada **project yang sama**, buatlah sebuah package baru bernama **interfacelatihan**.

Name and Location	
Package Name:	<input type="text" value="interfacelatihan"/>

2. Pada package yang baru dibuat tersebut, tambahkan sebuah **interface** baru dengan cara klik kanan pada package → **New** → **Java Interface...** Beri nama interface baru tersebut dengan nama **ICumlaude**.



3. Pada interface **ICumlaude** tersebut, tambahkan 2 *abstract methods* bernama **lulus()** dan **meraihIPKTinggi()**.

```
public interface ICumlaude
{
    public abstract void lulus();
    public abstract void meraihIPKTinggi();
}
```



4. Berikutnya, buatlah sebuah class baru bernama **Mahasiswa** dengan baris-baris kode seperti dibawah ini.

```
public class Mahasiswa
{
    protected String nama;

    public Mahasiswa(String nama)
    {
        this.nama = nama;
    }

    public void kuliahDiKampus()
    {
        System.out.println("Aku mahasiswa, namaku " + this.nama);
        System.out.println("Aku berkuliah di kampus.");
    }
}
```

5. Selanjutnya, buatlah class baru bernama **Sarjana** yang merupakan **turunan** dari class **Mahasiswa**. Class Sarjana tersebut dibuat meng-**implements** interface ICumlaude yang sudah dibuat sebelumnya tadi. Ketikkan kode di bawah pada class tersebut. **Tips:** Anda dapat menggunakan fasilitas *override* otomatis dengan cara yang sama yaitu dengan mengklik ikon lampu peringatan seperti pada percobaan 1.

```
public class Sarjana extends Mahasiswa implements ICumlaude
{
    public Sarjana(String nama)
    {
        super(nama);
    }

    @Override
    public void lulus() {
        throw new UnsupportedOperationException("Not supported yet.");
    }

    @Override
    public void meraihIPKTinggi() {
        throw new UnsupportedOperationException("Not supported yet.");
    }
}
```



6. Selanjutnya sesuaikan isi dari *method* **lulus()** dan **meraihIPKTinggi()** agar sama dengan baris kode di bawah.

```
public class Sarjana extends Mahasiswa implements ICumlaude
{
    public Sarjana(String nama)
    {
        super(nama);
    }

    @Override
    public void lulus()
    {
        System.out.println("Aku sudah menyelesaikan SKRIPSI");
    }

    @Override
    public void meraihIPKTinggi()
    {
        System.out.println("IPK-ku lebih dari 3,51");
    }
}
```

Perhatikan pada baris kode di atas, class **Sarjana** meng-**extend** class **Mahasiswa**, ini berarti, **Sarjana adalah Mahasiswa** sementara itu agar semua objek dari class **Sarjana** ini nantinya dapat disebut sebagai **Cumlaude** maka ia harus meng-**implements** interface **ICumlaude**.

7. Kemudian dengan **cara yang sama** buatlah class baru bernama **PascaSarjana** dengan baris kode seperti di bawah ini.

```
public class PascaSarjana extends Mahasiswa implements ICumlaude
{
    public PascaSarjana(String nama)
    {
        super(nama);
    }

    @Override
    public void lulus()
    {
        System.out.println("Aku sudah menyelesaikan TESIS");
    }

    @Override
    public void meraihIPKTinggi()
    {
        System.out.println("IPK-ku lebih dari 3,71");
    }
}
```



8. Lalu buatlah sebuah class baru bernama **Rektor**. Class ini adalah class yang memanfaatkan class-class Mahasiswa yang telah dibuat sebelumnya.

```
public class Rektor
{
    public void beriSertifikatCumlaude(ICumlaude mahasiswa)
    {
        System.out.println("Saya REKTOR, memberikan sertifikat cumlaude.");
        System.out.println("Selamat! silahkan perkenalkan diri Anda..");

        mahasiswa.lulus();
        mahasiswa.meraihIPKtinggi();

        System.out.println("-----");
    }
}
```

9. Terakhir, buatlah sebuah class baru bernama **Program** yang diletakkan pada **package yang sama** dengan class-class percobaan 2. Tambahlah baris kode berikut ini:

```
public class Program
{
    public static void main(String[] args)
    {
        Rektor pakRektor = new Rektor();

        Mahasiswa mahasiswaBiasa = new Mahasiswa("Charlie");
        Sarjana sarjanaCumlaude = new Sarjana("Dini");
        PascaSarjana masterCumlaude = new PascaSarjana("Elok");

        pakRektor.beriSertifikatCumlaude(mahasiswaBiasa);
        pakRektor.beriSertifikatCumlaude(sarjanaCumlaude);
        pakRektor.beriSertifikatCumlaude(masterCumlaude);
    }
}
```

10. Pada baris kode tersebut, apabila Anda mengetikkan semua class dengan benar, maka akan terdapat *error* dan class Program **tidak dapat** dieksekusi. Perbaikilah kode Anda agar program yang Anda buat mengeluarkan output seperti berikut ini:



```
Output - ilkom.uniwara.modul09 (run) x ICumlaude.java x
run:
Saya REKTOR, memberikan sertifikat cumlaude
Selamat! Silahkan perkenalkan diri Anda..
Aku sudah menyelesaikan SKRIPSI
IPK-ku lebih dari 3,51

-----
Saya REKTOR, memberikan sertifikat cumlaude
Selamat! Silahkan perkenalkan diri Anda..
Aku sudah menyelesaikan TESIS
IPK-ku lebih dari 3,71

-----
BUILD SUCCESSFUL (total time: 1 second)
```

11. Pertanyaan diskusi:

- Mengapa pada langkah nomor 9 terjadi error? Jelaskan!
- Dapatkan method **kuliahDiKampus()** dipanggil dari objek **sarjanaCumlaude** di class **Program**? Mengapa demikian?
- Dapatkan method **kuliahDiKampus()** dipanggil dari parameter **mahasiswa** di method **beriSertifikatCumlaude()** pada class **Rektor**? Mengapa demikian?
- Modifikasilah method **beriSertifikatCumlaude()** pada class **Rektor** agar hasil eksekusi class **Program** menjadi seperti berikut ini:

```
run:
Saya REKTOR, memberikan sertifikat cumlaude.
Selamat! Bagaimana Anda bisa cumlaude?
Aku mahasiswa, namaku Dini
Aku berkuliah di kampus.
Aku sudah menyelesaikan SKRIPSI
IPK-ku lebih dari 3,51

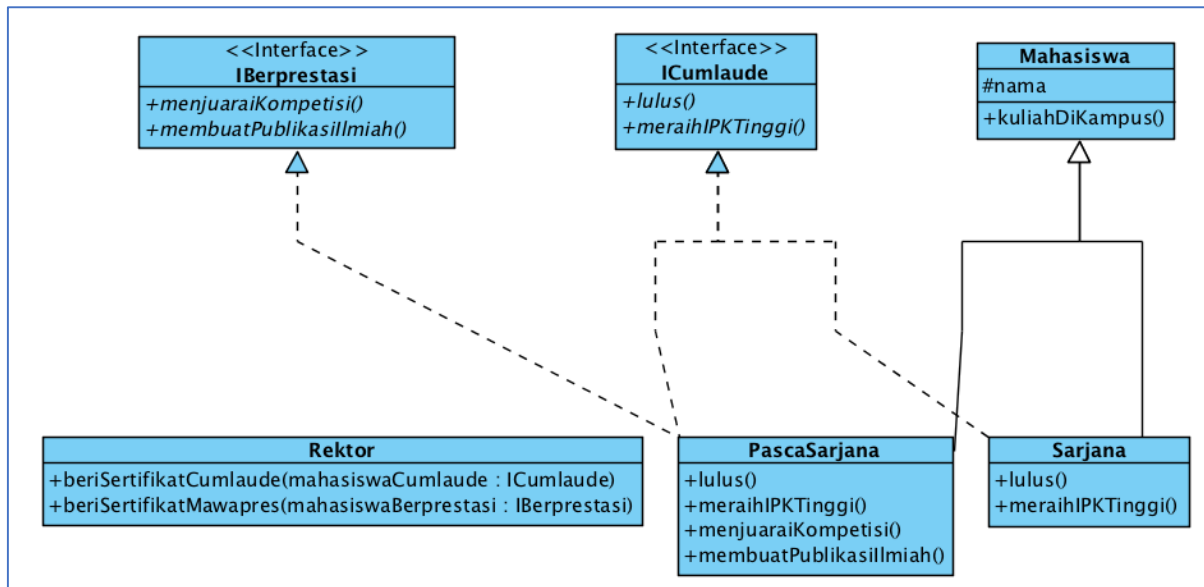
-----
Saya REKTOR, memberikan sertifikat cumlaude.
Selamat! Bagaimana Anda bisa cumlaude?
Aku mahasiswa, namaku Elok
Aku berkuliah di kampus.
Aku sudah menyelesaikan TESIS
IPK-ku lebih dari 3,71

-----
BUILD SUCCESSFUL (total time: 0 seconds)
```



Percobaan 3: Multiple Interfaces Implementation

Pada percobaan kali ini kita akan memodifikasi program yang telah dibuat pada Percobaan 2 sehingga pada program tersebut nantinya akan terdapat sebuah class yang *meng-implements* lebih dari 1 *interface*.



Bayangkan pada skenario sebelumnya, dimana seorang rektor juga akan **beriSertifikatMawapres()** pada sebuah acara wisuda. Mahasiswa yang berhak menerima penghargaan tersebut tentunya adalah mahasiswa yang berprestasi, dimana kriteria prestasi di sini berbeda antara mahasiswa Sarjana dengan mahasiswa Pasca Sarjana. Pada percobaan ini, kita akan menentukan kriteria prestasi yaitu: harus **menjuaraiKompetisi()** dan **membuatPublikasiIlmiah()**.

1. Pada package yang sama dengan package pada Percobaan 2, tambahkan sebuah *interface* baru yang bernama **IBerprestasi**. Tambahkan baris kode seperti berikut didalamnya.

```

public interface IBerprestasi
{
    public abstract void menjuaraiKompetisi();
    public abstract void membuatPublikasiIlmiah();
}

```

2. Selanjutnya, **modifikasilah** class **PascaSarjana** dengan menambahkan interface baru **IBerprestasi** dibelakang kata kunci **implements**. Lalu dengan cara yang sama seperti sebelumnya, kliklah ikon lampu peringatan untuk **meng-generate** semua method *abstract* dari interface **IBerprestasi** pada class **PascaSarjana**.

```

13 public class PascaSarjana extends Mahasiswa implements ICumlaude, IBerprestasi
    {

```



3. Modifikasilah *method* yang telah di-generate oleh NetBeans menjadi seperti berikut.

```
@Override
public void menjuaraiKompetisi() {
    System.out.println("Saya telah menjuarai kompetisi INTERNASIONAL");
}

@Override
public void membuatPublikasiIlmiah() {
    System.out.println("Saya menerbitkan artikel di jurnal INTERNASIONAL");
}
```

4. Tambahkan *method* **beriSertifikatMawapres()** dengan baris kode seperti di bawah, pada class **Rektor**.

```
public void beriSertifikatMawapres(IBerprestasi mahasiswa)
{
    System.out.println("Saya REKTOR, memberikan sertifikat MAWAPRES.");
    System.out.println("Selamat! Bagaimana Anda bisa berprestasi?");

    mahasiswa.menjuaraiKompetisi();
    mahasiswa.membuatPublikasiIlmiah();

    System.out.println("-----");
}
```

5. Terakhir, modifikasilah *method* **main()** pada class **Program** Anda. *Comment*-lah semua baris yang terdapat *method* **beriSertifikatCumlaude()**, lalu tambahkan baris kode baru seperti pada gambar di bawah ini.

```
public class Program
{
    public static void main(String[] args)
    {
        Rektor pakRektor = new Rektor();

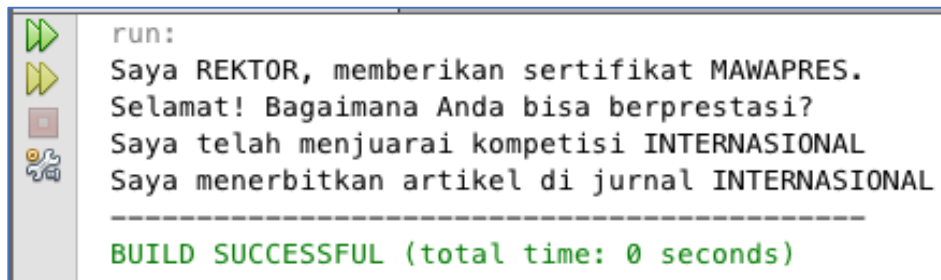
        //Mahasiswa mahasiswaBiasa = new Mahasiswa("Charlie");
        Sarjana sarjanaCumlaude = new Sarjana("Dini");
        PascaSarjana masterCumlaude = new PascaSarjana("Elok");

        //pakRektor.beriSertifikatCumlaude(mahasiswaBiasa);
        //pakRektor.beriSertifikatCumlaude(sarjanaCumlaude);
        //pakRektor.beriSertifikatCumlaude(masterCumlaude);

        pakRektor.beriSertifikatMawapres(sarjanaCumlaude);
        pakRektor.beriSertifikatMawapres(masterCumlaude);
    }
}
```



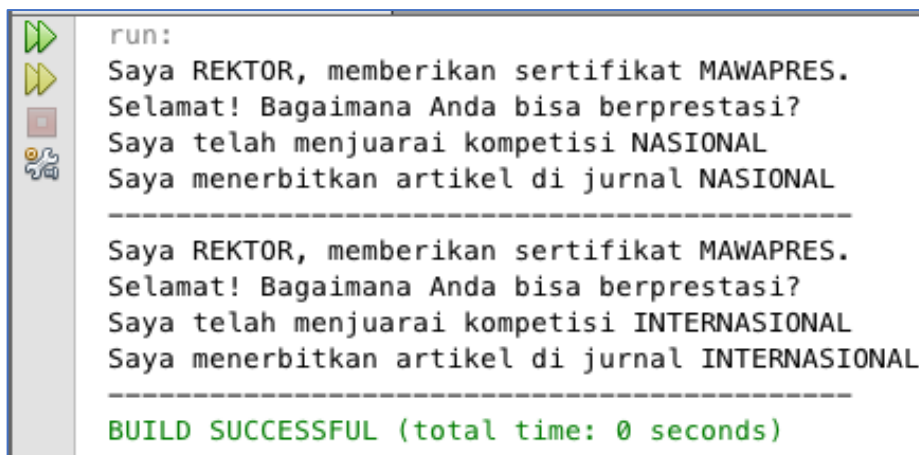

6. Akan terdapat *error* pada langkah-5, sehingga program tidak dapat dieksekusi. Perbaikilah kode programnya, sehingga hasil eksekusi menjadi **sama** seperti pada *screenshot* di bawah ini.



```
run:
Saya REKTOR, memberikan sertifikat MAWAPRES.
Selamat! Bagaimana Anda bisa berprestasi?
Saya telah menjuarai kompetisi INTERNASIONAL
Saya menerbitkan artikel di jurnal INTERNASIONAL
-----
BUILD SUCCESSFUL (total time: 0 seconds)
```

7. Pertanyaan diskusi:

Apabila **Sarjana Berprestasi** harus **menjuarai kompetisi NASIONAL** dan **menerbitkan artikel di jurnal NASIONAL**, maka modifikasilah class-class yang terkait pada aplikasi Anda agar di class **Program** objek **pakRektor** dapat memberikan sertifikat mawapres pada objek **sarjanaCumlaude**.



```
run:
Saya REKTOR, memberikan sertifikat MAWAPRES.
Selamat! Bagaimana Anda bisa berprestasi?
Saya telah menjuarai kompetisi NASIONAL
Saya menerbitkan artikel di jurnal NASIONAL
-----
Saya REKTOR, memberikan sertifikat MAWAPRES.
Selamat! Bagaimana Anda bisa berprestasi?
Saya telah menjuarai kompetisi INTERNASIONAL
Saya menerbitkan artikel di jurnal INTERNASIONAL
-----
BUILD SUCCESSFUL (total time: 0 seconds)
```