

ABSTRACTION

Fundamen Pengembangan Aplikasi
Pertemuan 12



UNIVERSITAS
ISLAM
INDONESIA

TOPIK MATERI

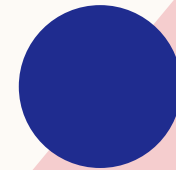
Abstraction

Abstract Class

Abstract Method

Interface

Interface vs Extends



OBJECT ORIENTED PROGRAMMING (OOP)

Empat konsep dasar OOP:

1. Encapsulation
2. Inheritance
3. **Abstraction**
4. Polymorphism



ABSTRACTION

ABSTRACTION

- *Abstraction* adalah proses menyembunyikan detail implementasi dan hanya menampilkan fungsionalitas kepada pengguna.
 - **Fokus** pada *apa yang dilakukan objek* **TIDAK** pada bagaimana melakukannya.
 - Misalnya, menyembunyikan seluruh **data yang tidak relevan**.
 - Dengan begitu, data yang akan **tampil** merupakan data yang *benar-benar menggambarkan suatu objek*.
- **Tujuan:** mengurangi kompleksitas
- Dua cara untuk melakukan *abstraction*, yaitu melalui:
 - *Abstract class*
 - *Interface*

ABSTRACTION

- Pada OOP, objek menyediakan abstraksi untuk *menyembunyikan detail* implementasi internalnya.
- Kita *hanya perlu* mengetahui apa *method* dari objek yang tersedia untuk dipanggil, apa saja input *parameter* yang dibutuhkan.
- Kita **tidak perlu** mengetahui bagaimana *method* tersebut diimplementasikan dan langkah apa yang dilakukan agar menghasilkan keluaran sesuai yang diharapkan.

SEBUAH ANALOGI

- Misalnya, Anda adalah objek yang sedang menunggu jemputan taksi online di bandara.
- Saat penjemputan, Anda harus memutuskan apa yang akan Anda katakan agar objek lain(driver) mengenalimu.
- Anda memutuskan untuk mengatakan beberapa informasi tentang Anda, seperti tinggi badan, warna kulit, **warna baju** yang Anda kenakan saat itu, dan **warna rambut**. *Beberapa informasi tersebut merupakan data yang akan membantu rencana pertemuan tadi.*
- **Di sisi lain**, ada banyak informasi tentang Anda yang perlu **disembunyikan** karena **tidak relevan** dengan situasi sebelum pertemuan tersebut.
 - Misalnya, Anda tidak perlu mengatakan **NIK** KTP-mu, **sifat**, dan semua informasi yang **tidak relevan** saat situasi Anda hendak bertemu dengan objek lain.
- Informasi-informasi **tidak relevan** tersebut **tidak akan membantu** objek lain menemukan Anda sebagai objek utama.
- Perlu diingat, pada sebuah objek **mungkin** memiliki *jumlah* **Abstraksi** yang sangat banyak. Oleh karena itu, **usahakan** untuk memilih informasi yang **benar-benar relevan** dan sesuai serta menggambarkan suatu objek.

ABSTRACT CLASS

The background features a large, light cream-colored circle on the left. To its right is a large, light pink circle. The top and bottom edges of the image are filled with a solid dark blue color. In the upper right corner, within the pink circle, there are several thin, white, concentric curved lines that fan out from the top edge.

ABSTRACT CLASS

- *Abstract class* didefinisikan sebagai *class* yang **TIDAK** bisa digunakan untuk membuat objek.
- Sebuah *class* agar dapat disebut *class* abstrak apabila setidaknya memiliki **satu** atau lebih *abstract method*
 - *Abstract method* adalah *method* yang **tidak** memiliki implementasi atau tidak ada bentuk konkritnya *alias* **tidak memiliki isi**.
- *Abstract class* hanya bisa digunakan sebagai **parent class**
 - Di dalamnya juga bisa memuat *abstract variabel* dan *abstract method* yang **WAJIB** *di-override* oleh kelas turunannya (*subclass*)
- *Subclass* yang memperluas (*atribute* dan *method*) dari *abstract class* ini **BISA** dibuatkan *object* namun masih tetap **memperhatikan modifier** yang digunakan di *abstract class* tersebut.

ABSTRACT METHOD

- Dideklarasikan dengan kata kunci **abstract**.
- Hanya *abstract class* yang dapat memiliki **abstract method**.
- Tidak memiliki blok statement atau *body method*.
- **SEMUA** **abstract method** yang ada di *abstract class* **HARUS** di-**override** oleh *subclass*.
- Isi atau *body method* dari **abstract method** didefinisikan di masing-masing *subclass*.

ABSTRACT METHOD

```
// ini abstrak method
void sayHello();

// ini bukan abstrak method karena
// punya implementasi di body method
void greeting(){
    System.out.println("Hello Java");
}
```

ABSTRACT METHOD

```
J Hewan.java X
src > J Hewan.java > ...
1  abstract class Hewan {
2      protected boolean apakahBisaBerenang;
3
4      String bernafas(){
5          return "Setiap hewan bernafas";
6      }
7
8      //abstract method
9      abstract String caraBernafas();
10 }
```

Karena berupa *abstract method* maka setiap *subclass* dari sebuah *abstract class* **HARUS** melakukan proses **override** *abstract method* tersebut.



```
J Ayam.java 1 X
src > J Ayam.java > Ayam
1  public class Ayam extends Hewan {
2      Ayam
3  }
4
The type Ayam must implement the inherited abstract method
Hewan.caraBernafas() Java(67109264)
View Problem (Alt+F8) Quick Fix... (Ctrl+.)
```

MENGAPA ABSTRACT CLASS?

- *Abstract class* memang **belum dapat digunakan secara langsung** sehingga harus dibuat bentuk konkritnya.
 - Caranya yaitu dengan membuat implementasi dari *method-method* yang masih abstrak melalui pewarisan (**inheritance**).
- *Subclass* akan membuat versi konkrit dari *abstract class*.
- *Lalu mengapa harus dibuat menjadi abstract class?*
- Pada kondisi tertentu, class induk (*super class*) **TIDAK** ingin kita buat sebagai objek karena kode *method*-nya belum jelas mau diimplementasikan seperti apa.

ABSTRACT CLASS

```
J Hewan.java X
src > J Hewan.java > ...
1  abstract class Hewan {
2      protected boolean apakahBisaBerenang;
3
4      String bernafas(){
5          return "Setiap hewan bernafas";
6      }
7
8      boolean bisaBerenang(){
9          return this.apakahBisaBerenang;
10     }
11
12     /*
13     * abstract method
14     */
15     abstract String caraBernafas();
16 }
```

```
J Ayam.java X
src > J Ayam.java > ...
1  public class Ayam extends Hewan {
2
3      @Override
4      String caraBernafas() {
5          return "Ayam bernafas dengan menggunakan paru-paru";
6      }
7
8      @Override
9      boolean bisaBerenang() {
10         super.apakahBisaBerenang = false;
11         return super.bisaBerenang();
12     }
13 }
```

```
J Ikan.java X
src > J Ikan.java > ...
1  public class Ikan extends Hewan {
2
3      @Override
4      String caraBernafas() {
5          return "Ikan bernafas dengan menggunakan insang";
6      }
7
8      @Override
9      boolean bisaBerenang() {
10         super.apakahBisaBerenang = true;
11         return super.bisaBerenang();
12     }
13 }
```

- Pada class **Hewan** kita tidak tahu bagaimana cara **ayam** nantinya bernafas
- Ketika ada *perubahan pada cara bernafasnya* kita cukup merubahnya di *method* implementasinya (class **Ayam**) bukan pada fungsi utamanya.
- Hal ini merupakan salah satu keuntungan dari konsep **abstraksi**.

CIRI-CIRI ABSTRACT CLASS

1. Dideklarasikan dengan kata kunci **abstract**.
2. Tidak dapat diinstansiasi.
3. Saat suatu *class* ingin mengakses suatu *abstract class* maka *class* tersebut harus:
 - a. Mengaksesnya melalui **subclass** dari *abstract class* tersebut.
 - b. Menjadi **subclass** dari *abstract class* tersebut.
4. Dapat memiliki *abstract method* maupun *non-abstract method*.

CONTOH ABSTRACT CLASS

Contoh:

1. Ingin dibuat suatu kelas yang dapat membuat objek berbagai jenis hewan.
2. Namun, karena sangat bervariasinya jenis hewan dan perilakunya, kelas hewan akan dibuat dengan mendeskripsikan **sifat** dan **perilaku umumnya** saja dan dibuat menjadi *abstract class*.

CONTOH ABSTRACT CLASS

1. Buat class **Hewan** di proyek baru.
2. Class **Hewan** memiliki atribut umur dan *method* bersuara()
3. Akses *modifier* atribut umur dibuat *protected* agar dapat diturunkan ke *subclass*.
4. Untuk membuat class **Hewan** menjadi *abstract class*, cukup ditambahkan kata kunci **abstract** di definisi *class*.

```
J Hewan.java X
src > J Hewan.java > ...
1  public abstract class Hewan {
2      protected int umur;
3
4      public String bersuara(){
5          return "Hewan bersuara ...";
6      }
7  }
```

CONTOH ABSTRACT CLASS

5. Kita coba untuk meng-instansiasi objek dari *class Hewan* di *method main()*
6. Muncul *error* karena *abstract class* **tidak dapat** di-instansiasi.
7. Salah satu cara agar dapat menggunakan *abstract class* adalah menjadi **subclass** dari *abstract class*

```
J Main.java 1 X
src > J Main.java > ...
1 public class Main {
  Run | Debug
2   public static void
3       Hewan hw = new Hewan();
4   }
5 }
```

Hewan
Cannot instantiate the type Hewan
View Problem (Alt+F8) No quick fixes available

8. Buat class **Kucing** yang mewarisi class **Hewan**.

```
src > J Kucing.java > ...
1 public class Kucing extends Hewan {
2
3 }
```

9. Coba meng-instansiasi class **Kucing**

```
src > J Main.java > ...
1 public class Main {
  Run | Debug
2   public static void main(String[] args) {
3       Kucing meow = new Kucing();
4       System.out.println(meow.bersuara());
5   }
6 }
```

PROBLEMS OUTPUT DEBUG CONSOLE

Hewan bersuara ...
PS D:\CODING\Abstraksi\Abstraksi>

CONTOH ABSTRACT CLASS

Contoh (melanjutkan contoh sebelumnya):

1. Jika diperhatikan *method* **bersuara()** terlalu umum.
2. Setiap jenis hewan memiliki perilaku bersuaranya masing-masing dan sebagian besar hewan mengeluarkan suara.
3. Agar setiap jenis hewan dapat memiliki perilaku bersuaranya masing-masing dan *memaksa* semua hewan memiliki perilaku bersuara maka *method* **bersuara()** dibuat menjadi **abstract method**.

CONTOH ABSTRACT CLASS

1. Method **bersuara()** di class **Hewan** sebelumnya.

```
src > J Hewan.java > ...
1  public abstract class Hewan {
2      protected int umur;
3
4      public String bersuara(){
5          return "Hewan bersuara ...";
6      }
7  }
```

2. Sekarang, *method* **bersuara()** di class **Hewan** dibuat menjadi *abstract method* dengan menambahkan kata kunci **abstract** didefinisi *method*

3. Perhatikan, muncul error karena *abstract method* **tidak boleh** memiliki *body method*.

src > J Hewan.java > ...	String Hewan.bersuara()
1 public abstract class Hewan {	Abstract methods do not specify a body
2 protected int umur;	View Problem (Alt+F8) Quick Fix... (Ctrl+.)
3	
4 public abstract String bersuara(){	
5 return "Hewan bersuara ...";	
6 }	
7 }	

4. Untuk menghilangkan error, *body method* dan *kurung kurawal* **dihapus** dari *method* **bersuara()**

```
src > J Hewan.java > ...
1  public abstract class Hewan {
2      protected int umur;
3
4      public abstract String bersuara();
5
6  }
```

CONTOH ABSTRACT CLASS

5. Saat *method* **bersuara()** di class **Hewan** diubah menjadi *abstract method* maka akan muncul *error* di class **Kucing**.

```
src > J Kucing.java > ...
1 public class Kucing extends Hewan {
2     Kucing
3 }
4
5 The type Kucing must implement the
6 inherited abstract method
7 Hewan.bersuara() Java(67109264)
View Problem (Alt+F8) Quick Fix... (Ctrl+.)
```

6. *Error* disebabkan karena class **Kucing** belum meng-*override* *abstract method* class **Hewan** yaitu *method* **bersuara()**

7. *Method* **bersuara()** di class **Kucing**.

```
src > J Kucing.java > ...
1 public class Kucing extends Hewan {
2     @Override
3     public String bersuara() {
4         return "Meong...meong...";
5     }
6 }
```

8. Jika kode di *method* **main()** dijalankan lagi

```
src > J Main.java > ...
1 public class Main {
2     Run | Debug
3     public static void main(String[] args) {
4         Kucing meow = new Kucing();
5         System.out.println(meow.bersuara());
6     }
}
```

PROBLEMS OUTPUT DEBUG CONSOLE

Meong...meong...

PS D:\CODING\Abstraksi\Abstraksi>

CONTOH ABSTRACT CLASS

1. Jika ingin menambah *subclass* lagi dari class **Hewan** maka *class* tersebut dapat memiliki perilaku bersuaranya sendiri juga.
2. Misalnya kita buat class **Burung**

```
src > J Burung.java > ...  
1 public class Burung extends Hewan {  
2     @Override  
3     public String bersuara() {  
4         return "Ciuiitt...ciuiitt";  
5     }  
6 }
```

3. Tambahkan kode di *method* **main()**

```
src > J Main.java > ...  
1 public class Main {  
2     Run | Debug  
3     public static void main(String[] args) {  
4         Kucing meow = new Kucing();  
5         Burung br = new Burung();  
6  
7         System.out.println("Suara kucing "+meow.bersuara());  
8         System.out.println("Suara burung "+br.bersuara());  
9     }  
}
```

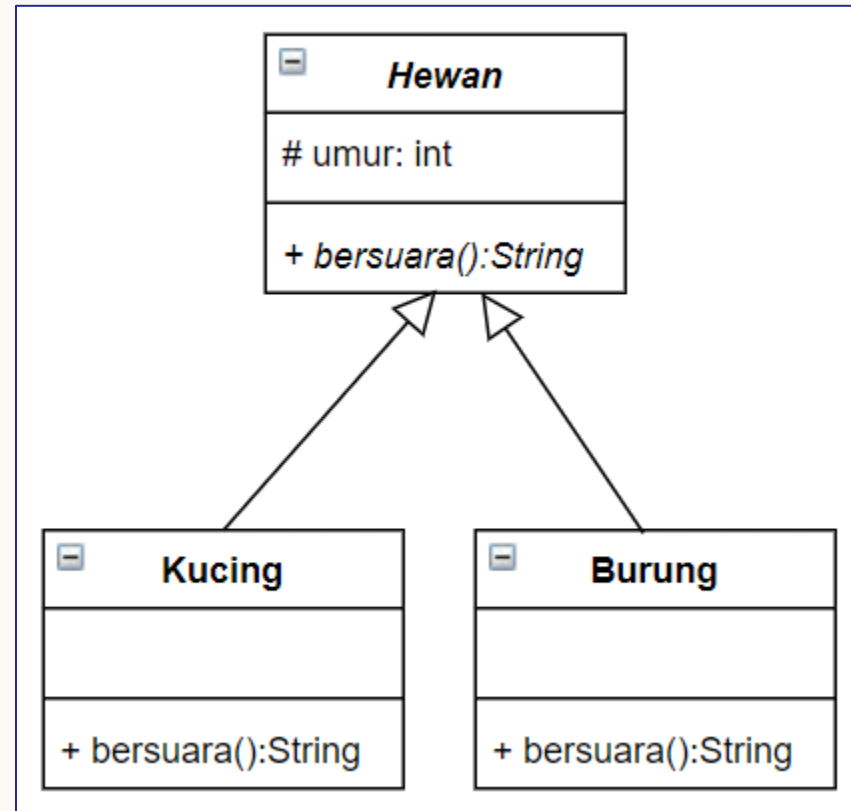
PROBLEMS OUTPUT DEBUG CONSOLE

```
Suara kucing Meong...meong...  
Suara burung Ciuiitt...ciuiitt  
PS D:\CODING\Abstraksi\Abstraksi>
```

4. Setiap *subclass* memiliki perilaku bersuaranya masing-masing.
5. Class **Hewan** hanya mendeskripsikan perilaku yang dimiliki *subclass*-nya

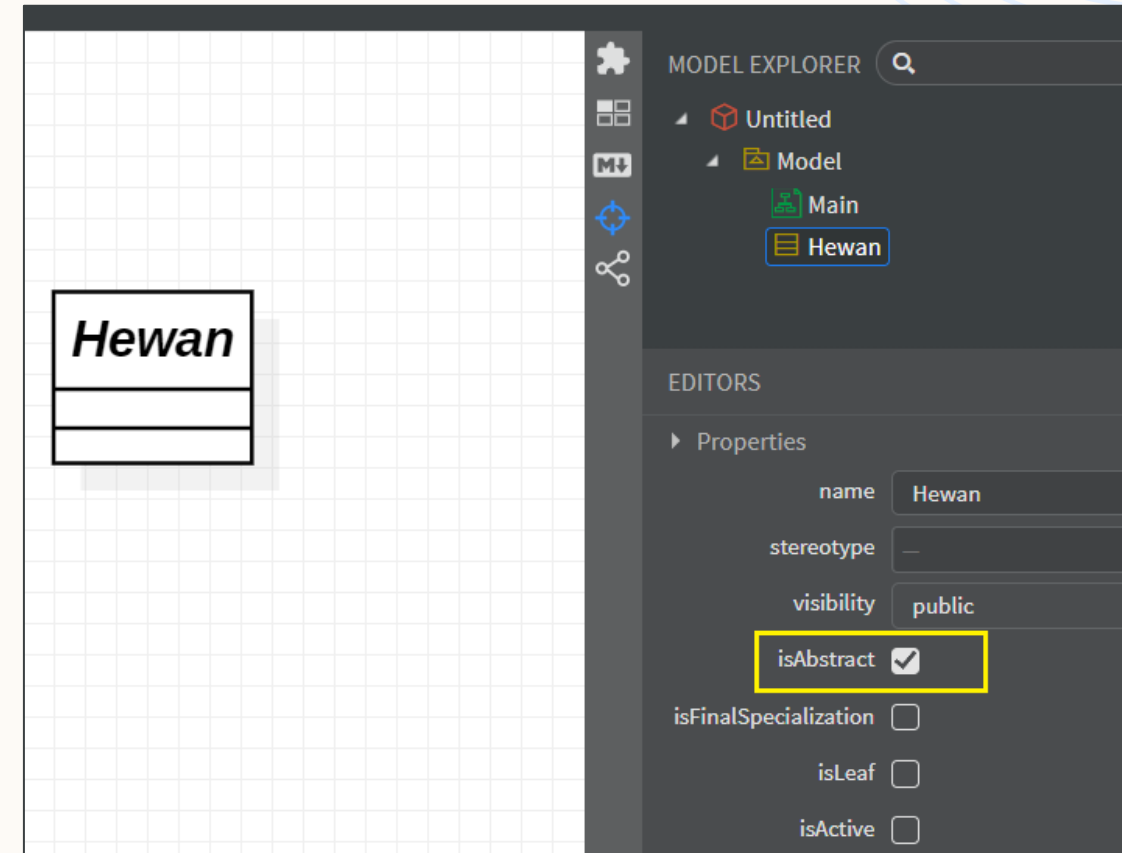
CLASS DIAGRAM ABSTRACT CLASS

1. Penulisan *abstract class* dan *method* sama seperti penulisan *class* dan *method* yang biasa.
2. Bedanya, penulisan *abstract class* dan *abstract method* ditulis dengan *cetak miring* atau *italic*.



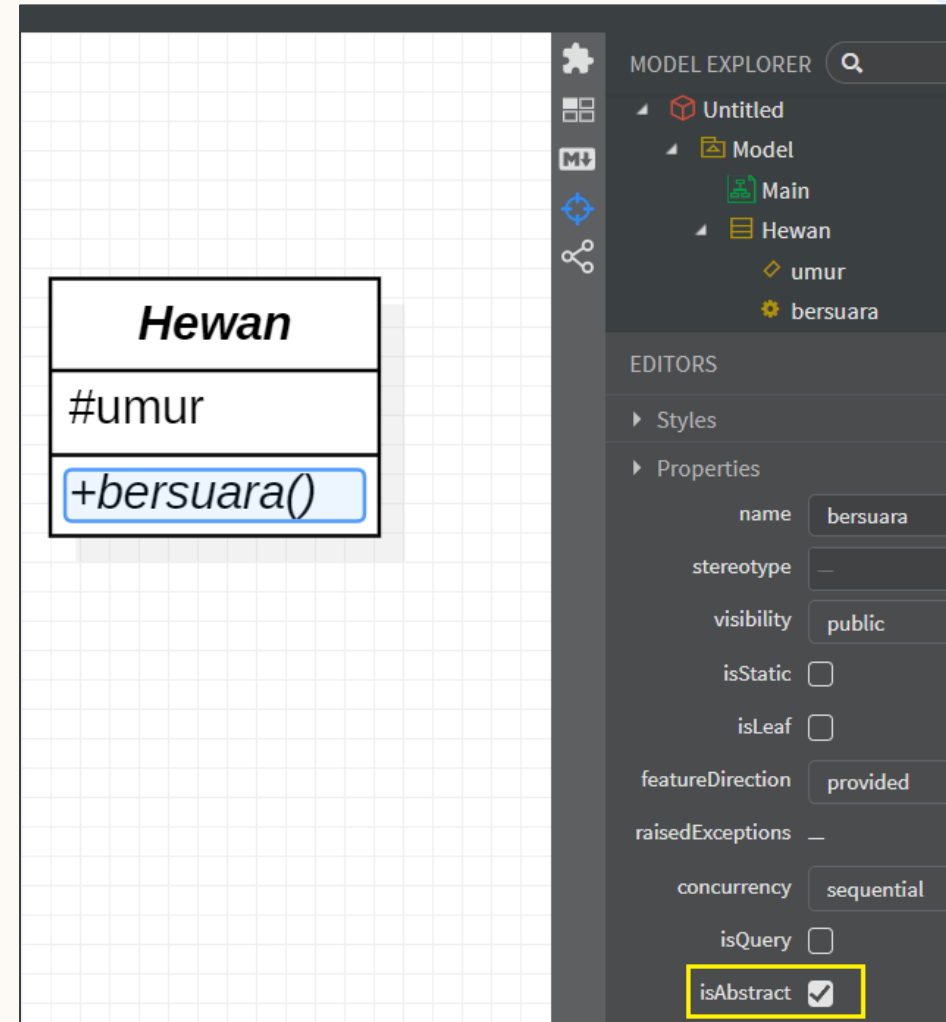
CLASS DIAGRAM ABSTRACT CLASS

- *Abstract class* dibuat mirip dengan *class* biasa
- Namun, pada properties **isAbstract** diberi tanda centang



CLASS DIAGRAM ABSTRACT CLASS

- Begitu juga ketika membuat *abstract method* di StarUML





INTERFACE

INTERFACE

- OOP memungkinkan kita untuk membuat *method* yang sifatnya **abstrak** yang belum terdefinisikan detilnya.
 - Tujuannya adalah untuk *menyederhanakan ide solusi dari permasalahan*.
- Keberadaan ***abstract method*** bisa terdapat dalam dua tempat, yaitu bisa terdapat dalam sebuah *class*, yang selanjutnya disebut dengan ***abstract class*** dan juga bisa terdapat dalam sebuah ***interface***.
- Interface adalah mekanisme yang *mirip* dengan sebuah *class* akan tetapi di dalam interface hanya terdiri dari kumpulan ***method kosong*** dan **KONSTANTA** saja.
- Interface **TIDAK** dapat dibuat obyek namun hanya dapat ***diimplementasi***.
- Interface ***mirip*** dengan *abstract class* dengan beberapa perbedaan

INTERFACE VS ABSTRACT CLASS

Abstract Class	Interface
<ul style="list-style-type: none">• Dapat mengandung <i>abstract method</i> maupun <i>method</i> biasa• Dapat mendeklarasikan variabel instan• Digunakan oleh kelas lain melalui pewarisan dengan kata kunci <i>extends</i>• Dibuat dengan menggunakan kata <i>class</i>	<ul style="list-style-type: none">• HANYA dapat mengandung <i>abstract method</i>• Hanya dapat mendefinisikan KONSTANTA saja• Digunakan oleh kelas lainnya melalui mekanisme implementasi dengan kata kunci <i>implements</i>• Dibuat dengan menggunakan kata <i>interface</i>, bukan <i>class</i>

IMPLEMENTASI INTERFACE

- Implementasi *interface* memiliki konsep yang mirip dengan *extends* pada *abstract class* yang memiliki aturan sebagai berikut:
 - Implementasi *interface* mewariskan **seluruh** *method* dan KONSTANTA.
 - Setiap kelas yang mengimplementasikan *interface* **WAJIB** meng-*override* dan mendeklarasikan ulang setiap *method*-nya agar kelas bersifat konkret.
 - Jika ada satu atau lebih *method* yang **TIDAK** di-*override* maka *interface* tersebut wajib didefinisikan sebagai *abstract class*.

SYARAT INTERFACE

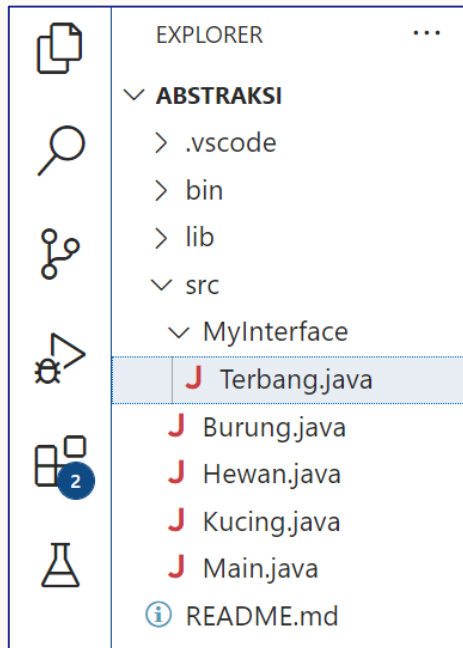
- **TIDAK** boleh ada *deklarasi konstruktor*
- Semua *method*, implisit maupun eksplisit, memiliki modifier **abstract**.
- Semua *attribute* memiliki modifier public, static, dan final (semua attribute bertindak sebagai KONSTANTA).
- Sebuah *class* yang mengimplemetasi sebuah interface harus meng-*override* setiap *method* yang ada pada interface tersebut.

CONTOH INTERFACE

- Akan dibuat suatu *interface* yang memiliki spesifikasi fungsional umum untuk terbang.
- Spesifikasi fungsional umum untuk terbang yang baru dibuat kali ini yaitu
 - Dapat menambah ketinggian, serta
 - Memberi batas ketinggian untuk dapat terbang.

CONTOH INTERFACE

1. Buat *package* baru kemudian buat *interface* **Terbang** di *package* baru tersebut.



2. Perhatikan bahwa pembuatan *interface* **tidak** menggunakan kata *class*

3. Interface **Terbang** memiliki *attribute* **KETINGGIAN_MAKS** dan *method* **tambahKetinggian()**

```
src > MyInterface > J Terbang.java > ...  
1  package MyInterface;  
2  
3  public interface Terbang {  
4      int KETINGGIAN_MAKS = 100;  
5  
6      int tambahKetinggian(int t);  
7  }
```

4. Perhatikan! Karena semua *attribute* di *interface* adalah konstanta maka *attribute* **KETINGGIAN_MAKS** ditulis dengan huruf besar semua.
5. Perhatikan bahwa *method* **tambahKetinggian()** **tidak** memiliki *body method*.

CONTOH INTERFACE

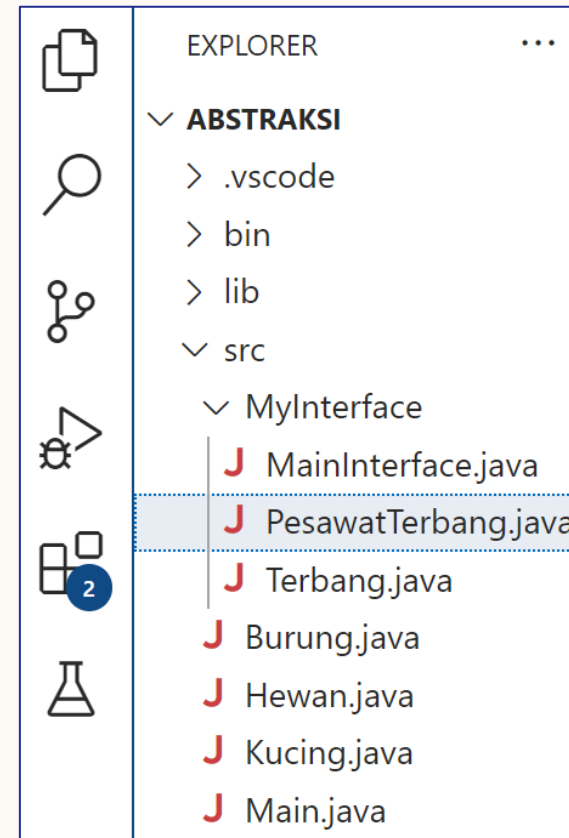
6. Perhatikan! Jika *method* **tambahKetinggian()** dipaksa memiliki *body method* maka akan muncul error.

```
src > MyInterface > J Terbang.java > ...  
1 package MyInterface;  
2  
3 public interface Terbang {  
4     int tambahKetinggian(int t)  
5  
6     int tambahKetinggian(int t){  
7         return t++;  
8     }  
9 }
```

Abstract methods do not specify a body
View Problem (Alt+F8) Quick Fix... (Ctrl+.)

7. Kembalikan *method* **tambahKetinggian()** ke kode awalnya yang tanpa *body method*.

8. Buatlah dua *class* baru yaitu *class* **PesawatTerbang** dan *class* yang punya *method* **main()**



CONTOH INTERFACE

9. Class **PesawatTerbang** akan meng-*implement* interface **Terbang**.

```
src > MyInterface > J PesawatTerbang.java > ...
1  package MyInterface;
2
3  public class PesawatTerbang implements Terbang {
4      The type PesawatTerbang must implement the inherited
5      abstract method
6      Terbang.tambahKetinggian(int) Java(67109264)
7
8      MyInterface.PesawatTerbang
9      View Problem (Alt+F8) Quick Fix... (Ctrl+.)
```

10. Perhatikan, ada *error*.
11. Error terjadi karena class **PesawatTerbang** belum meng-*override* semua *method* di interface **Terbang**.

12. Tambahkan *attribute* **ketinggian** dan *override* *method* **tambahKetinggian()** ke class **PesawatTerbang** lalu tambahkan isi dari *body* *method* **tambahKetinggian()**

```
src > MyInterface > J PesawatTerbang.java > ...
1  package MyInterface;
2
3  public class PesawatTerbang implements Terbang {
4      int ketinggian = 0;
5
6      @Override
7      public int tambahKetinggian(int t) {
8          ketinggian = ketinggian+t;
9          if(ketinggian < KETINGGIAN_MAKS){
10             return ketinggian;
11          }else{
12             return ketinggian = 100;
13          }
14      }
15  }
```

CONTOH INTERFACE

13. Buat objek dari class **PesawatTerbang** di *class* yang punya method **main()**
14. Serta panggil *method* **tambahKetinggian()**

```
src > MyInterface > J MainInterface.java > ...  
1  package MyInterface;  
2  
3  public class MainInterface {  
    Run | Debug  
4      public static void main(String[] args) {  
5          PesawatTerbang ps = new PesawatTerbang();  
6  
7          System.out.println(ps.tambahKetinggian(t:50));  
8          System.out.println(ps.tambahKetinggian(t:25));  
9          System.out.println(ps.tambahKetinggian(t:40));  
10     }  
11 }
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

50

75

100

PS D:\CODING\Abstraksi\Abstraksi>

PENGECEKAN CONTOH INTERFACE

1. Attribute **KETINGGIAN_MAKS** coba diakses langsung melalui *interface* **Terbang** untuk memastikan *attribute* tersebut memiliki kata kunci **static** secara implisit.

```
src > MyInterface > J MainInterface.java > MainInterface > main(String[])
1 package MyInterface;
2
3 public class MainInterface {
4     public static void main(String[] args) {
5         Terbang.
6         KETINGGIAN_MAKS : int      Terbang.KETINGGIAN_MAKS : int
7         class : Class<MyInterface.Terbang>
8     }
```

2. Perhatikan! Muncul rekomendasi *attribute* **KETINGGIAN_MAKS** di kode tersebut yang menandakan *attribute* **static**.

3. *Attribute* **KETINGGIAN_MAKS** coba diubah nilainya, untuk memastikan *attribute* tersebut sebuah **KONSTANTA** atau berkata kunci **final** secara implisit.

```
src > MyInterface > J MainInterface.java > ...
1 package MyInterf
2
3 public class Mai
4     public stati
5         Terbang.KETINGGIAN_MAKS = 900;
6
7 }
```

The final field Terbang.KETINGGIAN_MAKS cannot be assigned Java(33554512)

int KETINGGIAN_MAKS = 100

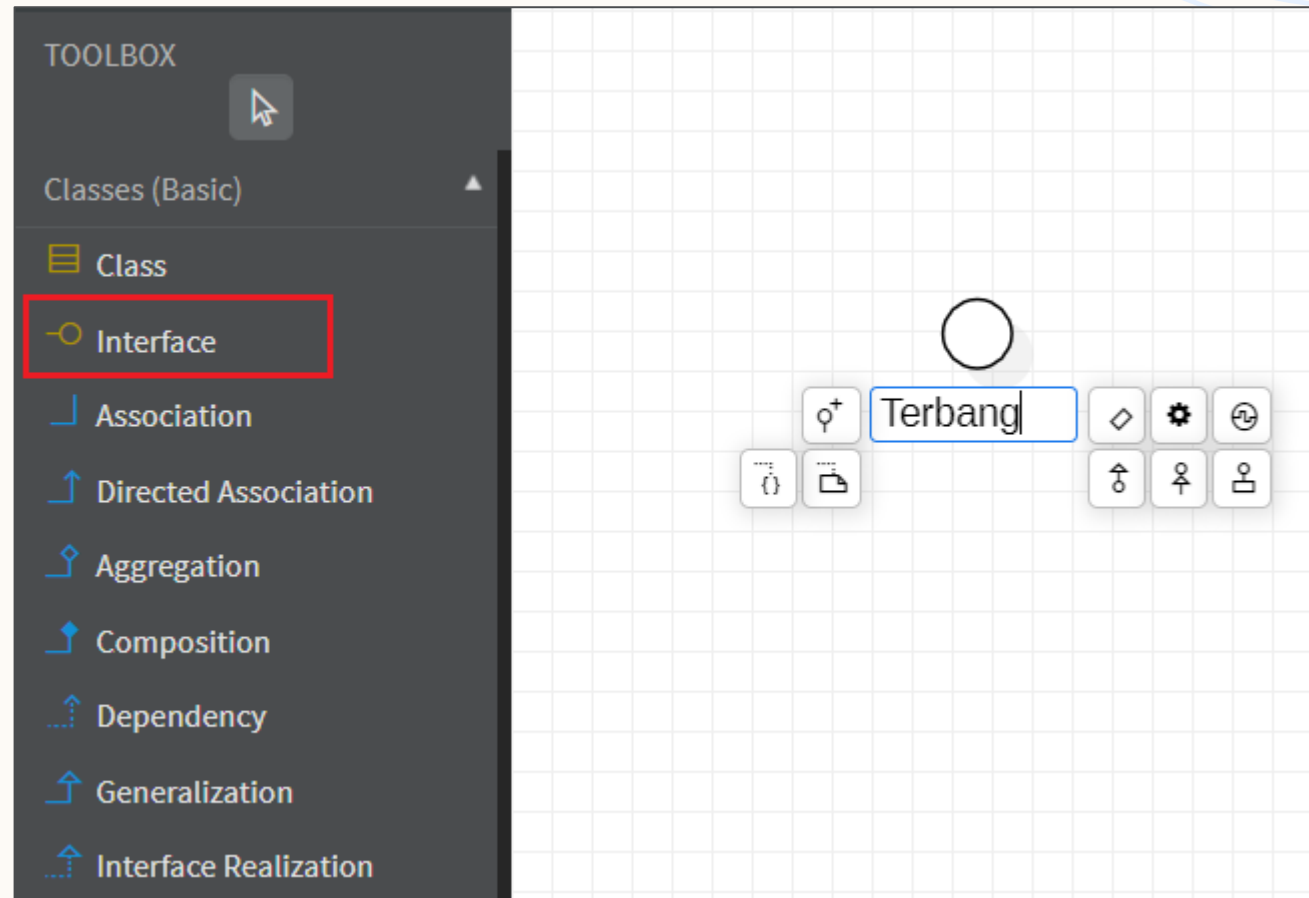
View Problem (Alt+F8) No quick fixes available

4. Muncul *error* yang menandakan *attribute* **final**.

CLASS DIAGRAM INTERFACE

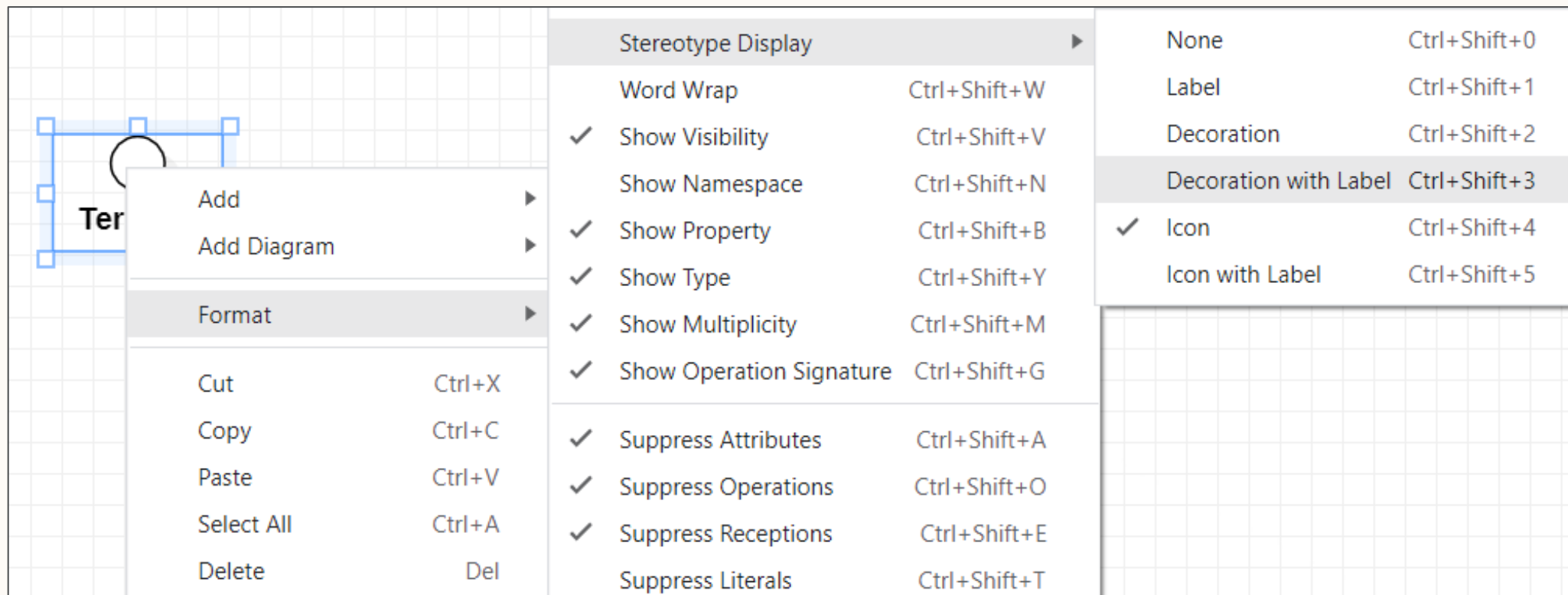
Berikut cara membuat gambar *interface* di aplikasi StarUML.

1. Pilih ikon **Interface** di bagian Toolbox
2. Klik di area utama aplikasi kemudian beri nama *interface*-nya
3. Secara *default* gambar *interface* berbentuk lingkaran



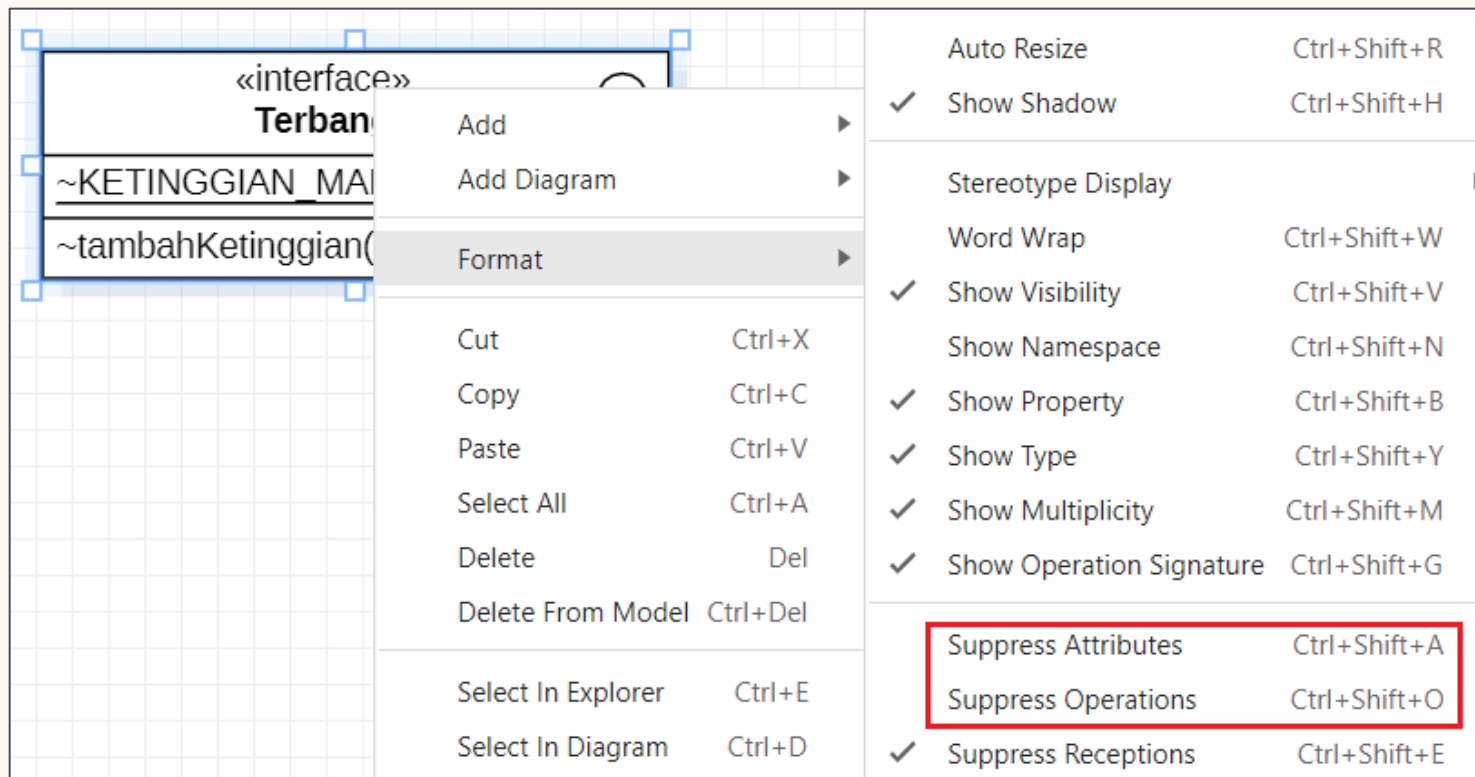
CLASS DIAGRAM INTERFACE

4. Jika gambar interface ingin terlihat kotak, klik kanan pada bagian interface lalu pilih **Format > Stereotype Display > Decoration with Label**



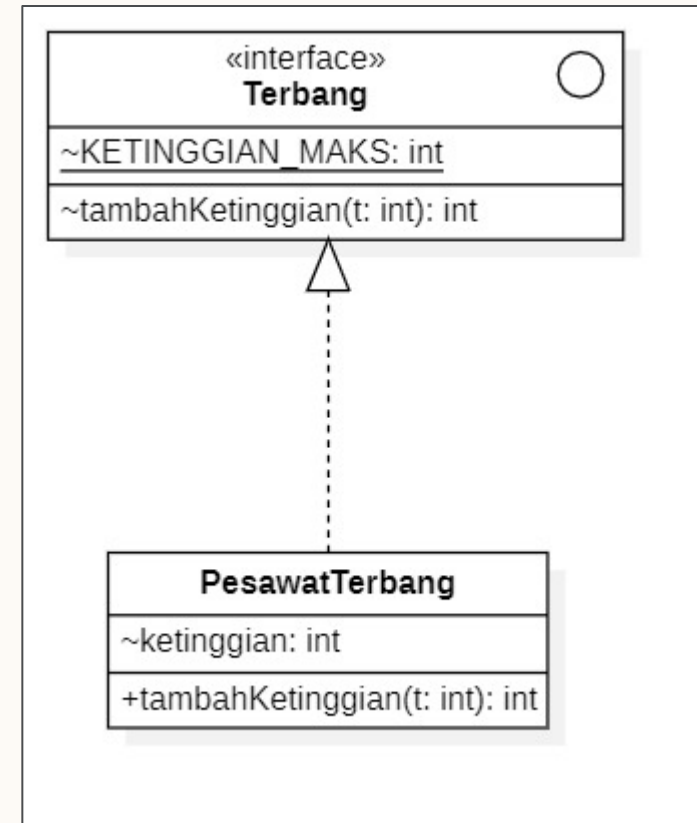
CLASS DIAGRAM INTERFACE

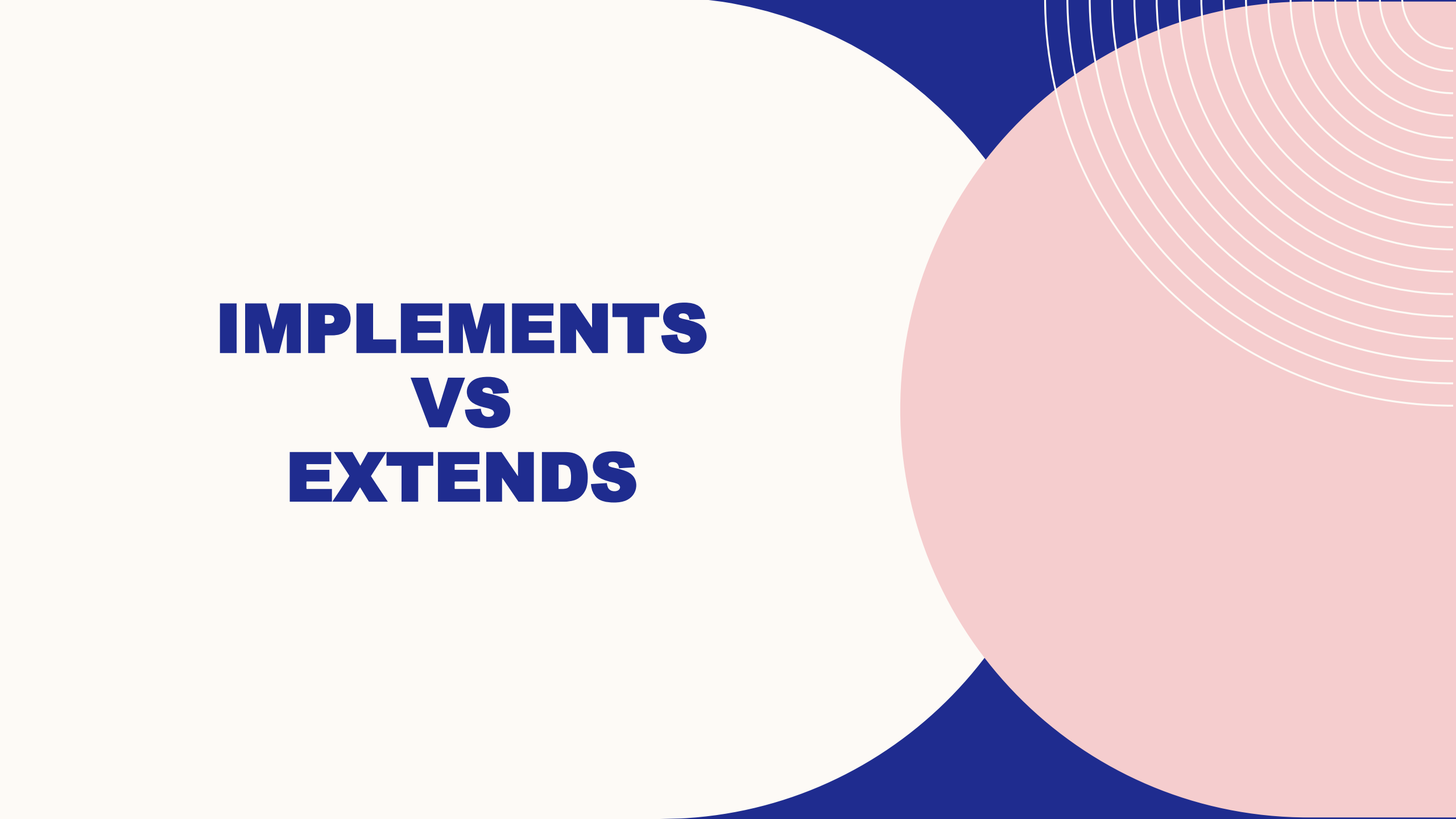
5. Tambahkan *attribute* dan *method* di *interface* yang sudah dibuat seperti pada class biasanya
6. Agar *attribute* dan *method* muncul di dalam gambar, pilih pengaturan **Format > Suppress Attributes** dan **Suppress Operations**



CLASS DIAGRAM INTERFACE

7. Gunakan relasi **Interface Realization** untuk proses *implements*
8. Hasil Akhir





IMPLEMENTS VS EXTENDS

IMPLEMENTS VS EXTENDS

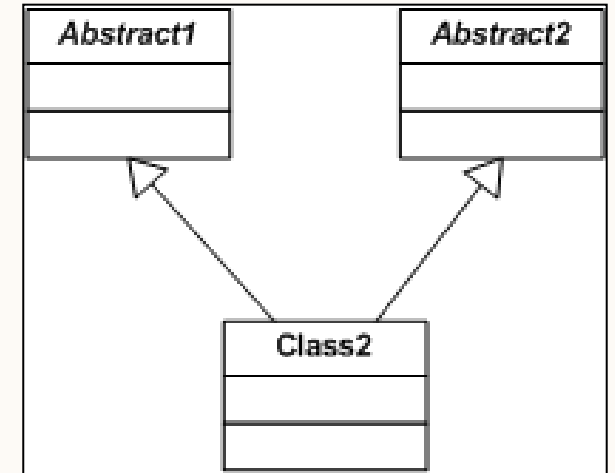
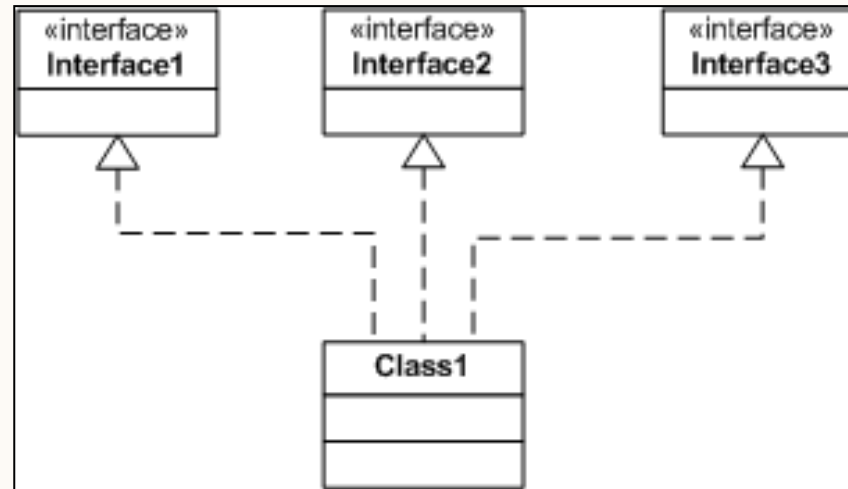
- **Extends** tidak memungkinkan adanya Pewarisan Jamak

```
src > J Burung.java > ...
1  public class Burung extends Hewan, Mamalia {
2      @Override
3      public String bersuara() {
4          return "Ciuiitt...ciuiitt";
5      }
6  }
7
```

- **Interface** memungkinkan adanya Pewarisan Jamak

```
src > J Burung.java > ...
1  import MyInterface.Terbang;
2
3  public class Burung extends Hewan implements Terbang {
4      @Override
5      > public String bersuara() { ...
8
9      @Override
10 > public int tambahKetinggian(int t) { ...
13 }
```

IMPLEMENTS VS EXTENDS



- **Extends** tidak memungkinkan adanya Pewarisan Jamak
- **Interface** memungkinkan adanya Pewarisan Jamak

TERIMA KASIH

