Secara garis besar, **interface** berfungsi sama seperti abstract class, yakni berisi kumpulan *signature method* yang harus di implementasikan. Setiap class yang menggunakan interface akan "dipaksa" membuat ulang method tersebut. Berikut perbedaan antara **abstract class** dengan **interface**:

1. Abstract class bisa menjadi parent bagi class-class lain, sedangkan interface tidak berada di dalam struktur hierarki class sehingga tidak memiliki hubungan *child-parent* sebagaimana abstract class.
2. Abstract class bisa berisi konstanta, property dan method "biasa" (selain abstract method). Sedangkan interface hanya bisa berisi *signature* method dan konstanta saja. Dengan kata lain, interface tidak bisa diisi dengan property dan method biasa.
3. Interface dibuat menggunakan keyword implement.

Tujuan penggunaan interface juga mirip seperti abstract class, yakni membuat desain hubungan logika antar class (hierarki class). Hanya saja interface lebih ditujukan untuk kasus dimana terdapat abstract method yang kurang pas jika ditempatkan ke dalam abstract class. Melanjutkan contoh class **Produk** kita sebelumnya, setiap produk seperti Televisi, MesinCuci dan LemariEs merupakan turunan dari abstract class Produk. Misalkan beberapa produk akan di ekspor keluar negeri, sehingga saya perlu method cekHargaUsd() untuk mengetahui harga setiap barang dalam dollar Amerika (USD), serta method cekNegara() untuk mengetahui negara mana tujuan ekspor dari produk tersebut. Method cekHargaUsd() dan cekNegara() ini menurut saya kurang pas jika ditempatkan ke dalam class Produk karena tidak semua produk akan di ekspor (hanya beberapa saja). Jika dibuat ke dalam class baru juga kurang tepat karena tidak masuk ke dalam struktur hierarki class. Saya ingin semua class hanya berasal dari turunan class Produk saja. Dalam kasus seperti ini, method cekHargaUsd() dan cekNegara() menjadi kandidat yang pas untuk sebuah **interface**. Terlebih saya ingin setiap class produk yang akan di ekspor **harus** memiliki method cekHargaUsd() dan cekNegara().
Berikut format dasar pembuatan interface ProdukEkspor:

```
interface ProdukEkspor {
  public function cekHargaUsd();
  public function cekNegara();
}
```

Interface dibuat menggunakan keyword interface yang diikuti dengan nama interface tersebut. Dalam contoh ini saya membuat interface dengan nama ProdukEkspor. Isi dari interface hanya bisa berupa *signature method*, yakni nama method serta parameter (jika ada). Implementasi dari method-method ini nantinya diserahkan kepada class yang menggunakan interface.
Agar sebuah class bisa menggunakan interface, tambahkan keyword implements setelah nama class. Sebagai contoh, jika saya ingin class Televisi menerapkan interface ProdukEkspor, maka kodenya adalah sebagai berikut:

```
class Televisi implements ProdukEkspor {

}
```

Dengan kode ini, di dalam class Televisi saya harus mendefinisikan ulang method cekHargaUsd() dan cekNegara() sebagaimana penulisan *signature* method dari interface ProdukEkspor. Jika tidak, PHP akan mengeluarkan pesan error. Berikut kode program lengkap pembuatan interface ProdukEkspor serta implementasi method cekHargaUsd() dan cekNegara() di dalam class Televisi:

```php
<?php
interface ProdukEkspor {
  public function cekHargaUsd();
  public function cekNegara();
}

class Televisi implements ProdukEkspor {
  public function cekHargaUsd(){
    return 185;
  }
  public function cekNegara(){
    return ["Singapura", "Malaysia","Thailand"];
  }
}

$produk01 = new Televisi();
echo $produk01->cekHargaUsd();
echo "<br>";
echo implode(", ",$produk01->cekNegara());
```

Hasil kode program:

```
185
Singapura, Malaysia, Thailand
```

Sama seperti *abstract method*, implementasi dari setiap method yang ada di dalam interface diserahkan sepenuhnya kepada class. Di dalam class Televisi saya harus membuat ulang method cekHargaUsd() dan cekNegara(), dimana method cekHargaUsd() mengembalikan angka 185 dan method method cekNegara() mengembalikan sebuah array ["Singapura","Malaysia","Thailand"]. Selanjutnya di baris 16 saya membuat object $produk01 dari class Televisi, kemudian menampilkan hasil pemanggilan $produk01->cekHargaUsd() dan $produk01->cekNegara(). Khusus untuk method cekNegara(), di baris 19 saya menggunakan fungsi implode() bawaan PHP untuk mengkonversi array menjadi string. Ini karena method cekNegara() mengembalikan sebuah array yang tidak bisa langsung di-echo. Sebagai percobaan, saya akan membuat class MesinCuci dan menggunakan interface ProdukEkspor, tapi tidak membuat ulang method tersebut:

18.interface_error.php

```php
1   <?php
2   interface ProdukEkspor {
3     public function cekHargaUsd();
4     public function cekNegara();
5   }
6
7   class MesinCuci implements ProdukEkspor {
8   }
```

Hasil kode program:

```
// Fatal error: Class MesinCuci contains 2 abstract methods and must
therefore be declared abstract or implement the remaining methods
(ProdukEkspor::cekHargaUsd, ProdukEkspor::cekNegara)
```

Hasilnya tampil pesan error karena class MesinCuci tidak mengimplementasikan ulang semua method yang ada di dalam interface ProdukEkspor.
Signature method yang ada di dalam interface juga harus memiliki hak akses public, tidak bisa di set sebagai private atau protected:

19.interface_visibility_error.php

```php
<?php
interface ProdukEkspor {
  private function cekHargaUsd();
  protected function cekNegara();
}
```

Hasil kode program:

```
// Fatal error: Access type for interface method ProdukEkspor::cekHargaUsd()
must be omitted
```

Atau kita bisa tidak menulis visibility sama sekali karena itu akan dianggap PHP sebagai public:

```php
<?php
interface ProdukEkspor {
  function cekHargaUsd();
  function cekNegara();
}
```

Sebuah class juga bisa mengimplementasikan banyak interface sekaligus. Berikut contohnya:

```php
<?php
interface ProdukEkspor {
  public function cekHargaUsd();
  public function cekNegara();
}

interface ProdukMakanan {
  public function cekExpired();
}

interface ProdukMakananBeku {
  public function cekSuhuMin();
}

class Nugget implements ProdukEkspor, ProdukMakanan, ProdukMakananBeku
{
  public function cekHargaUsd(){
    return 7.5;
```

```
18    }
19    public function cekNegara(){
20      return ["Singapura", "Malaysia","Thailand"];
21    }
22    public function cekExpired(){
23      return "April 2019";
24    }
25    public function cekSuhuMin(){
26      return -14;
27    }
28  }
29
30  $produk01 = new Nugget();
31  echo $produk01->cekHargaUsd();
32  echo "<br>";
33  echo implode(", ",$produk01->cekNegara());
34  echo "<br>";
35  echo $produk01->cekExpired();
36  echo "<br>";
37  echo $produk01->cekSuhuMin();
```

Hasil kode program:

```
7.5
Singapura, Malaysia, Thailand
April 2019
-14
```

Dalam kode program ini saya membuat 3 buah interface: ProdukEkspor, ProdukMakanan dan ProdukMakananBeku. Interface ProdukMakanan di rancang untuk produk makanan. Dimana nantinya saya perlu mengetahui kapan tanggal expired dari produk makanan tersebut. Oleh karena itu saya ingin semua produk makanan memiliki method cekExpired(). Interface ProdukMakananBeku di tujukan untuk produk makanan yang harus disimpan dalam suhu dingin. Oleh karena itu saya juga ingin setiap produk makanan beku memiliki method cekSuhuMin(). Method ini untuk mengetahui berapa suhu minimum penyimpanan makanan. Di baris 20 saya membuat class Nugget yang mengimplementasikan ketiga interface tersebut. Perhatikan cara penulisannya, yakni dengan keyword implements dan diikuti dengan ketiga nama interface (dipisah dengan tanda koma). Karena hal ini, di dalam class Nugget saya harus mendefinisikan ulang ke-4 method tersebut, yakni method cekHargaUsd() dan cekNegara() milik interface ProdukEkspor, method cekExpired() milik interface ProdukMakanan, serta method cekSuhuMin() milik interface ProdukMakananBeku. Untuk method cekHargaUsd() akan mengembalikan nilai 7.5 yang artinya harga produk ini adalah US$ 7.5. Method cekNegara() mengembalikan array 3 negara tujuan ekspor, yakni ["Singapura", "Malaysia","Thailand"]. Kemudian method cekExpired() mengembalikan string "April 2019" yakni tanggal expired produk Nugget. Terakhir method cekSuhuMin() mengembalikan angka -14, dimana produk ini bisa disimpan dalam suhu minimum -14 derajat celcius.

Interface Inheritance

Interface juga bisa diturunkan ke dalam interface lain. Caranya sama seperti penurunan class, yakni menggunakan keyword extends:

```php
<?php
interface ProdukEkspor {
  public function cekHargaUsd();
  public function cekNegara();
}

interface ProdukMakanan {
  public function cekExpired();
}

interface ProdukMakananBeku extends ProdukMakanan {
  public function cekSuhuMin();
}

class Nugget implements ProdukEkspor, ProdukMakananBeku {
  public function cekHargaUsd(){
    return 7.5;
  }
  public function cekNegara(){
    return ["Singapura", "Malaysia","Thailand"];
  }
  public function cekSuhuMin(){
    return -14;
  }
}
```

Hasil kode program:

```
Fatal error: Class Nugget contains 1 abstract method and must therefore be declared abstract or implement the remaining methods (ProdukMakanan::cekExpired)
```

Perhatikan di baris 11, saya membuat interface ProdukMakananBeku sebagai turunan dari interface ProdukMakanan. Hasilnya, interface ProdukMakananBeku akan memiliki signature method cekExpired() milik interface ProdukMakanan. Class Nugget hanya menggunakan interface ProdukEkspor dan ProdukMakananBeku saja (tidak menggunakan interface ProdukMakanan). Oleh kerena itu saya mencoba untuk tidak mengimplementasikan method cekExpired(), karena method tersebut milik interface ProdukMakanan.
Hasilnya PHP menampilkan pesan error "komplain" karena saya tidak membuat ulang method cekExpired(). Meskipun method ini berada di interface ProdukMakanan yang tidak dipakai class Nugget, namun method ini secara tidak langsung ada di dalam interface ProdukMakananBeku.

Interface Constant

Yang cukup unik dari interface adalah, kita diperbolehkan membuat konstanta:

22.interface_const.php

```php
<?php
interface ProdukEkspor {
  public function cekHargaUsd();
  public function cekNegara();
  public const biayaPajak = 0.5;
}

echo ProdukEkspor::biayaPajak;   // 0.5
```

Di baris 5 saya membuat konstanta biayaPajak dan memberinya nilai 0.5. Cara pengaksesannya ini mirip seperti konstanta class, yakni dengan menggunakan *scope resolution operator* berupa tanda " :: ". Masih berhubungan dengan class, penamaan interface di dalam PHP "berbagi tempat" dengan nama class. Maksudnya, tidak boleh ada nama class dan interface yang sama:

23.interface_naming.php

```php
<?php
interface ProdukEkspor {
  public function cekHargaUsd();
  public function cekNegara();
}

class ProdukEkspor{
}
```

Hasil kode program:

```
// Fatal error: Cannot declare class ProdukEkspor, because the name is
already in use
```

Di sini saya membuat interface bernama ProdukEkspor, kemudian membuat class dengan nama yang sama. Hasilnya terjadi error karena nama ProdukEkspor sudah dipakai oleh interface. Error seperti ini juga akan terjadi jika kita membuat 2 buah class menggunakan nama yang sama.

Menutup pembahasan kita tentang interface, pertanyaan cukup sering adalah *kapan harus menggunakan abstract class dan kapan harus menggunakan interface?* Sebenarnya tidak ada rumus baku karena ini lebih ke seperti apa logika anda untuk merancang kode program (design class yang dipakai). Tapi tips singkatnya, jika method tersebut hampir selalu dipakai di dalam class turunan, maka tempatkan ke dalam abstract class. Namun jika method tersebut relatif jarang terpakai dan tidak secara langsung berhubungan dengan struktur class, maka tempatkan ke dalam interface.