Typescript

1. Penulisan tipe data
    a. Deklarasi data menggunakan Type inference
        i.
```
let number=10;
```
        ii.
```
let employee = {
    firstName: 'John',
    lastName: 'Doe',
    age: 25,
    jobTitle: 'Web Developer'
};
```
        iii. Cek type data:
```
console.log(typeof(number));
console.log(typeof(employee));
```
    b. Deklarasi data menggunakan Type annotation
        i.
```
let nama:string='typescript';
```
,
```
let nama:string;
nama="typescript";
```
        ii.
```
let employee: {
    firstName: string;
    lastName: string;
    age: number;
    jobTitle: string;
};
```
,
```
employee = {
    firstName: 'John',
    lastName: 'Doe',
    age: 25,
    jobTitle: 'Web Developer'
};
```
,
```
let employee: {
    firstName: string;
    lastName: string;
    age: number;
    jobTitle: string;
} = {
    firstName: 'John',
    lastName: 'Doe',
    age: 25,
    jobTitle: 'Web Developer'
};
```
2. Tipe data
    a. Number:
```
let number:number=10; const phy:number=3.14
```
, tipe data number termasuk floating point, integer, bigint, binary,octal, hexadecimal

seperti berikut:

```
//Binari dimuali 0 diikuti B besar atau kecil(0b atau 0B) kemudian dikuti digit 0 atau 1=>0b100,0B010
let bin=0b100;
let bin2 :number=0B010;
//Octal dimulai 0 diikuti o kecil(0o)kemudian diikuti angka range 0-7=>0o10,0o13,0o27
let octal:number=0o71;
//Hexadecimal dimulai dari 0 diikuti X besar atau kecil(0X atau 0x)diikuti angka atau huruf dlm range 0123456
let hd:number=0XA;
//Big Integer mewakili integer lebih dari 2^53 -1, mempunyai karakter n dibelakang integer
let big: bigint = 9007199254740991n;
```

b. String :

```
let nama:string="typescript"; const secretKey:string="typescript12345"
```

c. Boolean: `let bools:boolean=1;` , error ditandai dengan gelombang merah dibawab bools ketika pemberian nilai 0 atau 1 pada tipe data boolean karena dianggap tipe number. Harusnya: `let bools:boolean=true;`

d. Any: bisa tipe data apapun, misalnya:

```
let result:any;

result=90;
result='test'
result=['a',1]
console.log(result);
```

e. Union : memberikan lebih dari 1 tipe data, misalnya:

```
let temp:string|number|boolean;
temp="typescript";
temp=10;
temp=false;
```

f. Alias: membuat tipe baru sebagai alias dari tipe union, misalnya:

```
type alphanumeric = string | number ;
let input: alphanumeric;
input = 100; // valid
input = 'Hi'; // valid
//input = false; // Compiler error
```

g. Array:

```
let arrayNameString=[]; //tidak bisa array kosong saat inisialisasi dianggap type never
let arrayNamenoType: []=[]; //Jika dideklarasi tanpa type maka tidak bisa assign value arraynya
```

Array number artinya array berisi tipe number saja, misalnya:

```
let arrayName:number[];
let arrayName2:number[]=[]; //deklarasi sebuah variabel tipe array number kosong
arrayName=[1,2]
arrayName[2]=3
arrayName.push(4) //menambahkan pada bagian akhir
arrayName.push(5) //menambahkan pada bagian akhir
arrayName.pop() //menghapus dari bagian akhir
arrayName.unshift(5) //menambahkan pada bagian depan
arrayName.unshift(6) //menambahkan pada bagian depan
arrayName.shift() //menghapus dari bagian depan
//arrayName=['c','d'] //akan error karena array hanya bisa number
console.log(arrayName.length);
console.log(arrayName);
console.log(typeof (arrayName));
console.log(typeof arrayName);
```

Array string artinya array berisi tipe string saja, misalnya:

```
let arrayNameString:string[];
arrayNameString=['c','d']
```

Array union artinya array berisi beberapa tipe, misalnya:

```
let arrayName3: (string|number)[]; //bisa string atau number mis:[1,2,'b','c',4,'r']
arrayName3=[1,2,'b','c',4,'r']
```

Array any artinya array berisi tipe apapun, misalnya:

```
let arrayNameAny: any[]=[]; //bisa assing tipe string,number,boolean, dll
arrayNameAny=[1,2,'b','c',4,'r',true,{a:10}]
```

Array object artinya array berisi tipe object, misalnya:

```
//let arrayNameObj:{}[]=[];// deklarasi sebuah variabel tipe array object kosong
let arrayNameObj:{}[];// deklarasi sebuah variabel tipe array object kosong
arrayNameObj=[{nama:'budi',usia:20},{nama:'wati',usia:23}]
console.log(arrayNameObj);
```

h. Object

```
let employee: {
    firstName: string;
    lastName?: string;
    age: number;
    jobTitle?: string;
} = {
    firstName: 'John',
    lastName: 'Doe',
    age: 25,
    jobTitle: 'Web Developer'
};
```

```
let employee: {
    firstName: string,
    lastName: string,
    age: number,
    jobTitle: string,
};
employee = {
    firstName: 'John',
    lastName: 'Doe',
    age: 25,
    jobTitle: 'Web Developer'
};
```

```
let employee: {};
employee = {
    firstName: 'John',
    lastName: 'Doe',
    age: 25,
    jobTitle: 'Web Developer'
};
```

```
let employee: object;
employee = {
    firstName: 'John',
    lastName: 'Doe',
    age: 25,
    jobTitle: 'Web Developer'
};
```

Object kosong dan object dideklarasi tanpa ada inisialisasi nilai awal:

```
let person: {}={};
let person1

console.log(person);//akan object kosong {}
console.log(person1);//akan undefined
```

i. Undefined: jika deklarasi tanpa ada nilai awal

```
let person1
console.log(person1);//akan undefined
```

```
let person2=undefined;
console.log(person2); //akan undefined
console.log(typeof(person2)); //akan undefined
```

j. Null

```
let person1=null;
console.log(person1); // akan null
console.log(typeof(person1)); //akan object
```

3. Operator
    a. Arithmetic

Operator	Sign
Addition	+
Subtraction	-
Multiplication	*
Division	/

    b. Assignment

Operator	Meaning	Description
`a = b`	`a = b`	Assigns the value of b to a.
`a += b`	`a = a + b`	Assigns the result of a plus b to a.
`a -= b`	`a = a - b`	Assigns the result of a minus b to a.
`a *= b`	`a = a * b`	Assigns the result of a times b to a.
`a /= b`	`a = a / b`	Assigns the result of a divided by b to a.
`a %= b`	`a = a % b`	Assigns the result of a modulo b to a.
`a &=b`	`a = a & b`	Assigns the result of a AND b to a.
`a |=b`	`a = a | b`	Assigns the result of a OR b to a.
`a ^=b`	`a = a ^ b`	Assigns the result of a XOR b to a.
`a <<= b`	`a = a << b`	Assigns the result of a shifted left by b to a.
`a >>= b`	`a = a >> b`	Assigns the result of a shifted right (sign preserved) by b to a.
`a >>>= b`	`a = a >>> b`	Assigns the result of a shifted right by b to a.

    c. Unary

Unary Operators	Name
+x	Unary Plus
-x	Unary Minus
++x	Increment Operator (Prefix)
–x	Decrement Operator (Prefix)
x++	Increment Operator (Postfix)
x–	Decrement Operator (Postfix)

d. Logical

a	b	a && b
true	true	true
true	false	false
false	true	false
false	false	false

a	b	a || b
true	true	true
true	false	true
false	true	true
false	false	false

e. Comparison

Operator	Meaning
===	strict equal
!==	not strict equal

Operator	Meaning
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
==	equal to
!=	not equal to

4. Control Flow Statement
    a. IF

```
const maxType: number = 10;
let counterMaxType: number = 6;

if (counterMaxType < maxType) {
    counterMaxType++

}

console.log(counterMaxType);
```

    b. Nested IF

```
let age = 16;
let state = 'CA';

if (state == 'CA') {
  if (age >= 16) {
    console.log('You can drive.');
  }
}
```

```
let age = 16;
let state = 'CA';

if (state == 'CA' && age == 16) {
  console.log('You can drive.');
}
```

c. IF..ELSE

```
const maxType: number = 10;
let counterMaxType: number = 6;

if (counterMaxType < maxType) {
    counterMaxType++

}else{
    counterMaxType--
}

console.log(counterMaxType);
```

d. IF..ELSE IF

```
let nilaiType: number = 65
if (nilaiType === 65) {
    console.log("Nilai yang sama");
}
else if (nilaiType > 65) {
    console.log("Nilai lebih besar");
}
else if (nilaiType >= 65) {
    console.log("Nilai lebih besar sama dengan");
}
else if (nilaiType < 65) {
    console.log("Nilai kurang dari");
}
else if (nilaiType <= 65) {
    console.log("Nilai kurang dari sama dengan")
}
else  {
    console.log("Nilai tidak sama dengan");
}
```

e. IF..IF

```
let nilaiType: number = 65
if (nilaiType === 65) {
    console.log("Nilai yang sama");
}
if (nilaiType > 65) {
    console.log("Nilai lebih besar");
}
if (nilaiType >= 65) {
    console.log("Nilai lebih besar sama dengan");
}
if (nilaiType < 65) {
    console.log("Nilai kurang dari");
}
if (nilaiType <= 65) {
    console.log("Nilai kurang dari sama dengan")
}
if (nilaiType !== 65) {
    console.log("Nilai tidak sama dengan");
}
```

f. Ternary

```
let resultType = counterMaxType < maxType ? counterMaxType++ : counterMaxType--;
console.log(resultType);
console.log(counterMaxType < maxType ? counterMaxType++ : counterMaxType--)
```

g. Swicth Case

```
let a:number|string=3;
switch (a) {

    case 1:
        a = 'one';
        break;
    case 2:
        a = 'two';
        break;
    default:
        a = 'not found';
        break;
}
console.log(`The value is ${a}`);
```

```
let nilai:number = 65
switch (true){
    case nilai>=90:
    console.log("Grade A");
    break;
    case nilai>=80 && nilai <90:
    console.log("Grade B+");
    break;
    case nilai>=70 && nilai <80:
    console.log("Grade B");
    break;
    case nilai>=60 && nilai <70:
    console.log("Grade C");
    break;
    case nilai>=50 && nilai <60:
    console.log("Grade D");
    break;
    default:
    console.log("Grade E");

}
```

h. ES 2020

```
let xy="B";
xy &&= xy+"A";
```
sama dengan
```
let xy="B";
if (xy) {
    console.log(xy+"A");
}
```
.

Jika variable disebelah kiri operator && adalah true artinya tidak kosong maka akan melakukan proses selanjutnya jika tidak maka tidak melakukan apa-apa

5. Fungsi

Fungsi pada Typescript memiliki 2 type yaitu type parameter dan type output.
Sintaksnya sebagai berikut:

```
function nama(parameter: type, parameter:type,...): returnType {
    // do something
}
```

Penulisan fungsi dapat mengembalikan return atau void(tidak ada return).
Fungsi mengembalikan return,sebgai berikut:

```
function tambahReturn(a: number, b:number) :number{
    // do something
    return a+b
    //console.log(a+b);
}
```

Cara panggil fungsi: console.log(tambahReturn(10,20))

Fungsi Void, sebagai berikut:

```
function tambahVoid(a: number, b:number):void{
    // do something
    console.log(a+b);
}
```

Cara panggil fungsi: tambahVoid(10,20)

Jika type outuput adalah any, maka bisa return ataupun void, contoh:

```
function tambahAnyOutput(a: number, b:number):any {
    // do something
    return a+b
    //console.log(a+b);
}
```

Cara panggil fungsi: console.log(tambahAnyOutput(10,20))

```
function tambahAnyOutput(a: number, b:number):any {
    // do something
    //return a+b
    console.log(a+b);
}
```

Cara panggil fungsi: tambahAnyOutput(10,20)

Jika tidak mendeklarasikan type output maka dianggap type any, contoh:

```
function tambahReturn(a: number, b:number){
    // do something
    return a+b
    //console.log(a+b);
}
```

Cara panggil fungsi: console.log(tambahReturn(10,20))

Contoh penulisan fungsi output beberapa type atau union:

```
function tambahAny(a:any, b:any):number|string {
    // do something
    return a+b
    //console.log(a+b);
}
```

Cara panggil fungsi: console.log(tambahAny(10,20))

Atau menggunakan alias type, contoh:

```
type stringAngka = string | number;
function tambahAny(a:any, b:any):stringAngka {
    // do something
    return a+b
    //console.log(a+b);
}
```

Cara panggil fungsi: console.log(tambahAny(10,20))

Penulisan dalam funggsi Arrow:

```
const tambah=(a:number, b:number): number =>{
    // do something
    return a+b
}
```

6. Parameter
    a. Optional Parameter, hanya bisa dilakukan pada typescript dan optional parameter harus diakhir.
    Contoh 1

```
function multiply1(a?: number, b?: number, c?: number): number {
    if (typeof c !== 'undefined' && typeof b !== 'undefined' && typeof a !== 'undefined') {
        //if (c) {
        return a * b * c;
    }
    //return a * b;
    return 0;
}
console.log("hasil kali :"+multiply1());
```

Contoh 2

```
function multiply2(a: number, b: number, c?: number): number {

    if (!c) { //c===undefined
        return a * b;
    }
    return a * b * Number(c);
}

console.log("Hasil kali 2:"+multiply2(5,6));
```

b. Default Parameter

Aturan penggunaan default parameter adalah pada optional parameter dan rest parameter tidak boleh diberi nilai default.

Contoh 1

```
function applyDiscountType(price = 0, discount = 0.05) {
    return price * (1 - discount);
}
```

```
console.log(applyDiscountType(100, 0.25));
console.log(applyDiscountType(100)); // 95
console.log(applyDiscountType(100,undefined)); // 95
console.log(applyDiscountType(undefined, undefined)); // 0
console.log(applyDiscountType()); // 0
```

Contoh 2

```
function applyDiscount(price:number, discount:number = 0.05) {
    return price * (1 - discount);
}
```

c. Rest Parameter

Rest parameter akan selalu array. Pengunaan rest parameter tidak boleh lebih dari satu, hanya ada satu rest parameter yang bisa digunakan pada fungsi dan posisi harus pada bagian akhir parameter.

```
function getTotal(a: number, b: number, ...numbers: number[]): number {
    let total = 0;
    numbers.forEach((num) => total += num);
    return total + a + b;
    //return total;
}
console.log(getTotal(4, 3, 1, 2, 3, 4, 5, 6));
```

```
console.log(getTotal(10, 100)); // 110
console.log(getTotal(10, 20)); // 30
console.log(getTotal(10, 20, 30)); // 60
```

7. Built in function/Fungsi bawaan sistem
    a. Fungsi Array
    b. Fungsi Matematika
    c. Fungsi String
    d. Fungsi Tanggal dan waktu

8. Perulangan/iterasi

```
let ranks=['A','B','C','D','E']
for(let i=0;i<ranks.length;i++){
  console.log(ranks[i]);
}

for(let x of ranks){
    console.log(x);
}

for(let x in ranks){
    console.log(ranks[x]);
}

ranks.forEach(e => {
    console.log(e);
    });

ranks.map(e => {
    console.log(e);
});

let i=0;
while(i<ranks.length){
    console.log(ranks[i]);
    i++
}
```

Output:

```
A
B
C
D
E
```

Looping array object:

```
let ranks = [
    { nama: 'Andi', nilai: 80 },
    { nama: 'Budi', nilai: 70 },
    { nama: 'Ani', nilai: 90 },
    { nama: 'Wati', nilai: 75 },
    { nama: 'Indra', nilai: 90 }]
for (let i = 0; i < ranks.length; i++) {
    console.log(ranks[i]);
    console.log(ranks[i].nama,' ',ranks[i].nilai);
    console.log(ranks[i]['nama'],' ',ranks[i]['nilai']);
}
```

```
for (let x of ranks) {
    console.log(x);
    console.log(x.nama,' ',x.nilai);
    console.log(x['nama'],' ',x['nilai']);
}

for (let x in ranks) {
    console.log(ranks[x]);
    console.log(ranks[x].nama,' ',ranks[x].nilai);
    console.log(ranks[x]['nama'],' ',ranks[x]['nilai']);
}

ranks.forEach(e => {
    console.log(e);
    console.log(e.nama, ' ', e.nilai);
    console.log(e['nama'], ' ', e['nilai']);
});

ranks.map(e => {
    console.log(e);
    console.log(e.nama, ' ', e.nilai);
    console.log(e['nama'], ' ', e['nilai']);
});
```

```
let i = 0;
while (i < ranks.length) {
    console.log(ranks[i]);
    console.log(ranks[i].nama,' ',ranks[i].nilai);
    console.log(ranks[i]['nama'],' ',ranks[i]['nilai']);
    i++
}
```

Output:

```
{ nama: 'Andi', nilai: 80 }
Andi   80
Andi   80
{ nama: 'Budi', nilai: 70 }
Budi   70
Budi   70
{ nama: 'Ani', nilai: 90 }
Ani   90
Ani   90
{ nama: 'Wati', nilai: 75 }
Wati   75
Wati   75
{ nama: 'Indra', nilai: 90 }
Indra   90
Indra   90
```

9. Error Handling

   Menangkap error bawaan dari sistem:

```
try {
    let result = add(4, 5)
    console.log(result);
} catch (err) {
    //console.log(err); bisa cara seperti ini
    console.log(err.name, ":", err.message);
}
console.log("Silahkan coba lagi");
```

   Membuat custom error handling:

   Contoh menggunakan throw

```
try {
    let umur = '';
    if (umur == "")

        throw ("Validasi Error, Tidak boleh kosong")
    if (isNaN(umur))
        throw ("Validasi Error, Tidak number/angka")
    console.log(umur);

}
catch (err) {

    console.log(err);//cara panggil jika hanya throw

}
finally {
    console.log("Selalu diproses masuk try atau catch");
}
```

Contoh menggunakan throw new Error akan disimpan by sistem kedalam object(name dan message):

```
try {
    let umur = '';
    if (umur == "")

        throw new Error("Validasi Error, Tidak boleh kosong")
    //throw ("Validasi Error, Tidak boleh kosong")
    if (isNaN(umur))
        throw new Error("Validasi Error, Tidak number/angka")
        //throw ("Validasi Error, Tidak number/angka")
    console.log(umur);

}
catch (err) {

    //console.log(err);//cara panggil jika hanya throw saja
    console.log(err.name,':',err.message);//cara panggil jika hanya throw new error

}
finally {
    console.log("Selalu diproses masuk try atau catch");
}
```

Menggunakan custom error handling global:

```
const CustomHelper = (name, message) => {
    errName = name;
    errMsg = message;
    //return errName,":",errMsg;
    return { name: errName, message: errMsg };
}
```

```
try {
    let umur = '';
    if (umur == "")

        throw CustomHelper("Validasi Error", "Tidak boleh kosong")
    if (isNaN(umur))
        throw CustomHelper("Validasi Error", "Tidak number/angka")
    console.log(umur);

}
catch (err) {
    console.log(err.name, ":", err.message);
    console.log(err);

}
finally {
    console.log("Selalu diproses masuk try atau catch");
}
```

10. OOP

    Secara default access modifier properties atau atribut dan method pada class akan public jika tidak di deklarasikan.

    a. Class tanpa constructor

        i. Class

```
// person class
class PersonClass {
    //properties atau atribut
    firstname: string = '';
    lastnmae: string = '';
    age: number = 0
    static alamat: string = '';

    //method
    getFullName() {
        return `${this.firstname} ${this.lastnmae}`;
    }

}
```

        ii. Object

Access modifier static baik pada properties atau method tidak dapat diakses melalui object melainkan langsung dari nama kelas. Method static hanya dapat mengakses properties atau atribut yang static.

```
// create a new object of person
const person1 = new PersonClass()
person1.age = 20
person1.firstname = "Belajar"
person1.lastname = "typescript"
PersonClass.alamat="Jakarta"

console.log(person1);
console.log(person1.getFullName(),PersonClass.alamat);
```

```
// person class
class PersonClass {
    //properties atau atribut
    firstname: string = '';
    lastname: string = '';
    age: number = 0
    static alamat: string = '';

    //method
    getFullName() {
        return `${this.firstname} ${this.lastname} ${PersonClass.alamat}`;
    }

}

// create a new object of person
const person1 = new PersonClass()
person1.age = 20
person1.firstname = "Belajar"
person1.lastname = "typescript"
PersonClass.alamat="Jakarta"

console.log(person1);
console.log(person1.getFullName());
```

b. Class menggunakan Constructor
    i. Class

```
class Person {
    ssn: string;
    firstName: string;
    lastName: string


    constructor( ssn:string,   firstName:string,   lastName:string) {
        this.ssn = ssn;
        this.firstName = firstName;
        this.lastName = lastName;
    }
   get getFullNameSsn() {
        return `${this.firstName} ${this.lastName} ${this.ssn}`;
    }

   getFullName() {
        return `${this.firstName} ${this.lastName}`;
    }
}
```

Atau bisa cara kedua:

```
class Person {


    constructor(public ssn:string, public firstName:string, public lastName:string) {

    }
   get getFullNameSsn() {
        return `${this.firstName} ${this.lastName} ${this.ssn}`;
    }

   getFullName() {
        return `${this.firstName} ${this.lastName}`;
    }
}
```

ii. Object

```
let personObj = new Person('171-28-0926','John','Doe');
console.log(personObj.getFullNameSsn); //pemanggilan getter
console.log(personObj.getFullName()); //pemanggilan method
```

c. Inheritance/pewarisan

Pewarisan adalah pembuatan subclass atau juga sering disebut child class. Yang mewariskan disebut parent class atau super class. Semua properties sesuai access modifier pada parent atau super class akan diturunkan ke juga ke child atau sub class.

Contoh parent class/super class:

```
class Transportasi {


    constructor(public nama: string, public jlhRoda: number, public bahanBakar: string) {

    }


    getDataKendaraan(): string {
        return `Nama kendaraan:${this.nama}, jumlah roda: ${this.jlhRoda}, dengan bahan bakar ${this.bahanBakar}`
    }

    getBBKendaraan(): string {
        return `Bahan bakar kendaraan:${this.bahanBakar}`
    }
}
```

Contoh child class/sub class:

```
class TransportasiDarat extends Transportasi {

    static str='hello'
    static count=0
    constructor( nama:string,jlhRoda:number,bahanBakar:string,public warna: string) {
       super(nama, jlhRoda, bahanBakar)
        TransportasiDarat.count++
    }

    static getBB(): string {
        return this.str
    }

    getDetailInfo() {
        return `${super.getDataKendaraan()} dan berwarna ${this.warna} ${TransportasiDarat.str}`
    }

    getBBInfo() {
        return `${this.bahanBakar}`
    }

}
```

Membuat object dari class TransportasiDarat:

```
console.log(TransportasiDarat.getBB())
let mobil1 = new TransportasiDarat('Toyota', 4, 'Bensin', 'putih')
let mobil2 = new TransportasiDarat('Toyota', 4, 'Solar', 'Hitam')
let mobil3 = new TransportasiDarat('Toyota', 4, 'Premium', 'Silver')

console.log(TransportasiDarat.count)
console.log(mobil1.getDataKendaraan()); //method pada parent
//console.log(mobil1.getBB());
TransportasiDarat.str='Javascript'
console.log(TransportasiDarat.getBB());
console.log(mobil1.getDetailInfo());
console.log(mobil1.getBBInfo());
```

d. Encapsulation/access modifier
    i. Public: properties atau method dapat diakses dari manapun (class itu sendiri, class turunan, dan object)
    ii. Private:properties atau method hanya dapat diakses pada class itu sendiri (tidak dapat dari class turunan atau object)

```
class PersonPrivate {


    constructor(private ssn: string, private firstName: string, private lastName: string) {

    }

    getFullName(): string {
        return `${this.firstName} ${this.lastName} ${this.ssn}`;
    }
}
let personPrivate = new PersonPrivate('153-07-3130', 'John', 'Doe');
//console.log(personPrivate.ssn); // compile error
```

iii. Protected: properties atau method hanya dapat diakses pada class itu sendiri dan turunannya(dari object tidak bisa)

```
class PersonProtected {
    static status:boolean=true;
    protected ssn: string;
    protected firstName: string;
    protected lastName: string;


    constructor(ssn: string, firstName: string, lastName: string) {
        this.ssn = ssn;
        this.firstName = firstName;
        this.lastName = lastName;
    }

    getFullName(): string {
        return `${this.firstName} ${this.lastName}`;
    }
}
console.log(PersonProtected.status);
let personProtected = new PersonProtected('153-07-3130', 'John', 'Doe');
//console.log(personProtected.ssn); //compiled error krna hanya bisa diakses di class itu sendiri dan turunannya
```

iv. Static: hanya dapat diakses meggunakan nama classnya

e. Interface

```
//interface sebagai kumpulan properties atau method2 kosong
interface Mailable {
    // send(): any
    // queue(): any
    //later():any
    send(email: string): any
    queue(email: string): any


}
```

Interface implement ke class:

```
class Mail implements Mailable {
    // constructor(public email:string){

    // }
    // send(): any {
    //     console.log(`Sent email to ${this.email} `);
    //     //return true;
    // }
    // queue(): any {
    //     console.log(`Queue an email to ${this.email}.`);
    //     //return true;
    // }

    send(email: string): any {
        console.log(`Sent email to ${email} `);
        //return true;
    }
    queue(email: string): any {
        console.log(`Queue an email to ${email}.`);
        //return true;
    }
}
```

Membuat child dari interface:

```
interface FutureMailable extends Mailable {
    later(email: string, after: number): boolean
}
```

```
class MailFuture extends Mail implements FutureMailable {
    later(email: string, after: number): boolean {
        console.log(`Send email to ${email} in ${after} ms.`);
        return true;
    }
    // send(): any {
    //     console.log(`Sent email to ${email}. `);
    //     return true;
    // }
    // queue(): any {
    //     console.log(`Queue an email to ${email}.`);
    //     return true;
    // }
}
```

Jika tanpa extends dari parent class:

```
class MailFuture implements FutureMailable {
    later(email: string, after: number): boolean {
        console.log(`Send email to ${email} in ${after} ms.`);
        return true;
    }
    send(email: string): boolean {
        console.log(`Sent email to ${email}. `);
        return true;
    }
    queue(email: string): boolean {
        console.log(`Queue an email to ${email}.`);
        return true;
    }
}
```

f. Abstract

Mirip interface, bedanya dalam abstract terdapat method abstract yaitu method kosong dan ada juga method yang bisa dipanggil. Abstract class harus diwariskan tidak bisa dibuat object langsung dari abstract class.

Contoh:

```
abstract class EmployeeAbstract  {
    constructor(private firstName: string, private lastName: string) {
    }
    abstract getSalary(): number

    get fullName(): string {
        return `${this.firstName} ${this.lastName}`;
    }
    compensationStatement(): string {
        //return `${this.fullName()} makes ${this.getSalary()} a month.`;//jika getter diubah jadi method
        return `${this.fullName} makes ${this.getSalary()} a month.`;
    }

}
```

Interface bisa diimplement ke dalam abstract class, contoh:

```
interface Mailable {
    send(email: string): boolean
    queue(email: string): boolean

}
abstract class EmployeeAbstract implements Mailable {
    constructor(private firstName: string, private lastName: string) {
    }
    abstract getSalary(): number

    get fullName(): string {
        return `${this.firstName} ${this.lastName}`;
    }
    compensationStatement(): string {
        //return `${this.fullName()} makes ${this.getSalary()} a month.`;//jika getter diubah jadi method
        return `${this.fullName} makes ${this.getSalary()} a month.`;
    }
    public send(email: string): boolean {
        return true
    }
    public queue(email: string): boolean {
        return false
    }
}
```

Membuat class baru sebagai turunan dari abstract class, contoh:

```
class Contractor extends EmployeeAbstract {
    constructor(firstName: string, lastName: string, private rate: number, private hours: number) {
        super(firstName, lastName);
    }
    getSalary(): number {
        return this.rate * this.hours;
    }
}
```

g. Polymorphism

Banyak bentuk yang dibedakan dalam 2 bagian yaitu: overload dan override.

Overload: isi method sama untuk beberapa method(nama methodnya sama parameternya yang berbeda-beda), contoh:

```
class CounterClass {
    private current: number = 0;
    count(): number;
    count(target: number): number[];
    count(target?: any): any | any[] {
        if (target) {
            let values: number[] = [];
            for (let start = this.current; start <= target; start++) {
                values.push(start);
            }
            return values;
        }
        return ++this.current;
    }
}
let counter_class = new CounterClass();
console.log(counter_class.count()); // return a number
console.log(counter_class.count(20)); // return an array
```

Override: isi method berbeda-beda untuk satu method yang sama, artinya terdapat duplikat method tapi isinya berbeda-beda, contoh:

```
class PersonClass {
    constructor(private firstName: string, private lastName: string) {
    }

    getFullName(): string {
        return `${this.firstName} ${this.lastName}`;
    }
    describe(): string {
        return `Hello This is ${this.firstName} ${this.lastName}.`;
    }
}

class EmployeeClass extends PersonClass {
    constructor(firstName: string,lastName: string,private jobTitle: string) {
        super(firstName, lastName);
    }

    describe(): string {
        return (`${super.describe()} ` +
            super.getFullName() + ` I'm a ${this.jobTitle}.`);
    }
}

let employee_class = new EmployeeClass('John', 'Doe', 'Web Developer');
console.log(employee_class.describe());
```

11. Pengelolaan output

    Return array:

```
const outputArray=()=>{
    let arrStr=['Java','Script'];
    //console.log(arrStr);
    return arrStr
}
console.log(outputArray())
const strArr=outputArray()
console.log(strArr);
console.log(strArr.toString());
const [x,y]=outputArray()
console.log(x,y);
console.log(strArr[0]);
```

    Return string:

```
const outputString = () => {
    //console.log('Hi Javascript');
    let msg = 'Hi Javascript';
    return msg
}

console.log(outputString());
```

Return Object:

```
const outputObject = () => {
    let status = "Ok", msg = "Berhasil tambah data"
    let result = {
        'statusRes': status,
        'msg': msg
    }
    //console.log(result);
    // return{
    //     statusRes:status,
    //     msg:msg
    // }
    return result
}
//outputObject()
console.log(outputObject());
const {statusRes,msg}=outputObject()
console.log(statusRes, msg);
```

12. Asynchronous
    a. Proses asynchronous

```
const getNomorAntri = (nomor) => {
    setTimeout(() => {
        console.log(nomor);
        return nomor;
    }, 3000);
}
```

```
const pilihPaket = (nomor, paket) => {
    setTimeout(() => {
        try {

            if (nomor <= 0 || isNaN(nomor)) {
                throw "Silahkan antri";
            }

            if (paket === 'A') {
                console.log("KFC Paket A");
                return "KFC Paket A";
            } else {
                console.log("KFC Paket B");
                return "KFC Paket B";
            }

        }
        catch (err) {
            console.log(err);
            return err;
        }
    }, 2000);

}
```

```
const tagihan = (paket) => {
    setTimeout(() => {
        if (paket === 'KFC Paket A') {
            console.log(25000);
            return 25000;
        } else if (paket === 'KFC Paket B') {
            console.log(24000);
            return 24000;
        } else {
            console.log('Belum ada pemesanan')
            return 'Belum ada pemesanan'
        }
    }, 1000);

}
```

```
let nomor = getNomorAntri(10)
console.log(nomor);
let paket = pilihPaket(nomor, 'A')
console.log(paket);
let bill = tagihan(paket)
console.log(bill);
```

Yang akan keluar hasilnya lebih dulu adalah proses yang lebih cepat selesai dikerjakan.

b. Asynchronous menjadi synchronous dengan callback. Fungsi callback adalah fungsi yang parameternya sebuah fungsi, sehingga terjadi fungsi dipanggil dalam fungsi.

```
const getNomorAntri = (nomor, cb) => {
    setTimeout(() => {
        setTimeout(() => {
            console.log(nomor);
            let paket = cb(nomor, 'B')
            console.log(paket);
            setTimeout(() => {
                let bill = tagihan(paket)
                console.log(bill);
            }, 1000);
        }, 2000)
    }, 5000);
}
```

```
const pilihPaket = (nomor, paket) => {
    try {

        if (nomor <= 0 || isNaN(nomor)) {
            throw "Silahkan antri";
        }

        if (paket === 'A') {
            return "KFC Paket A";
        } else {
            return "KFC Paket B";
        }

    }
    catch (err) {
        return err;
    }

}
```

```
const tagihan = (paket) => {
    if (paket === 'KFC Paket A') {
        //console.log(25000);
        return 25000;
    } else if (paket === 'KFC Paket B') {
        //console.log(24000);
        return 24000;
    } else {
        //console.log('Belum ada pemesanan')
        return 'Belum ada pemesanan'
    }

}



getNomorAntri(-2, pilihPaket)
```

c. Asynchronous menjadi synchronous dengan promise chaining

```
function getNomorAntriChaining(nomor) {

    // const result= new Promise((resolve,reject)=>{
    //     setTimeout(()=>{
    //         resolve(nomor)
    //     },3000)
    // })

    // return result;
    return new Promise((resolve, reject) => {
    //const result = new Promise((resolve, reject) => {

        setTimeout(() => {
            resolve(nomor);
        }, 5000);

    })
    //return result
}
```

```
function pilihPaketChaining(nomor, paket) {
    //return new Promise((resolve, reject) => {
    const result = new Promise((resolve, reject) => {

        setTimeout(() => {

            if (nomor <= 0 || isNaN(nomor)) {
                reject(new Error("Silahkan antri"));
                // reject("Silahkan antri");

            }
            if (paket === 'A') {
                resolve("KFC Paket A");

            } else {
                resolve("KFC Paket B");

            }

        }, 2000);
    });
    return result

}
```

```
function tagihanChaining(paket) {
    //return new Promise((resolve, reject) => {
    const result = new Promise((resolve, reject) => {

        setTimeout(() => {

            if (paket === 'KFC Paket A') {
                resolve(25000)
            } else if (paket === 'KFC Paket B') {
                resolve(24000)
            } else {
                resolve('Belum ada pemesanan')
            }

        }, 1000);
    });
    return result
}
```

```
const result=getNomorAntriChaining(-10)
  result
    .then((nomor) => {
        console.log(nomor);
        return pilihPaketChaining(nomor, 'B')
    })
    .then((paket) => {
        console.log(paket);
        return tagihanChaining(paket)
    })
    .then((bill) => {
        console.log(bill);
    })
    .catch((err) => {
        console.log(err.name,err.message);
    })
```

d. Asynchronous menjadi synchronous dengan async dan await (return yang dibungkus dalam promise adalah syarat untuk menggunakan async dan await). Beberapa method bawan library seperti untuk ajax dan database sudah return promise.

   Penulisan method sama seperti promise chaining, namun cara prosesnya pakai async dan await:

```
async function orderKFC (nomor) {
//const orderKFC = async (nomor) => {
    try {

        const no_urut = await getNomorAntriPromise(nomor);
        console.log(no_urut);
        const paket = await pilihPaketPromise(no_urut, 'B');
        console.log(paket);
        const bill = await tagihanPromise(paket)
        console.log(bill);
    }
    catch (err) {
        console.log(err.toString());
    }
}

orderKFC(25)
```