OOP

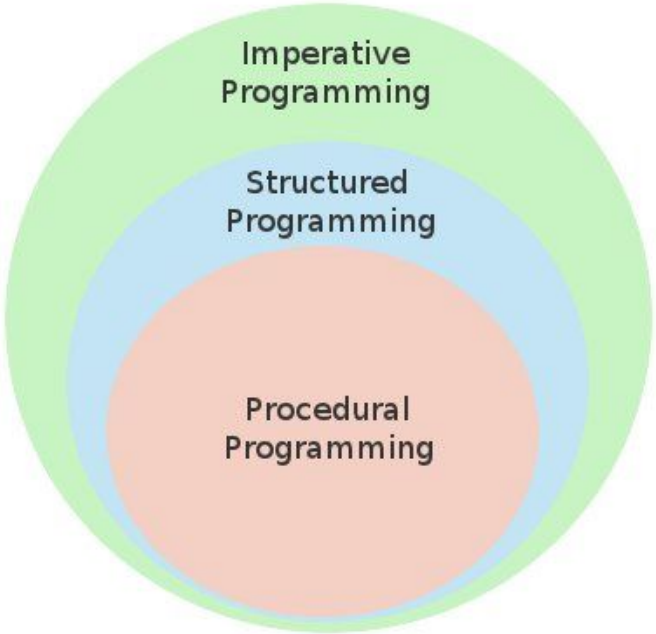
Isumi
Batam, Jan 2020

GOALS

1. Programming Paradigm
2. OOP Concept
3. Abstraction
4. Encapsulation
5. Inheritance
6. Polymorphism

1. Programming Paradigm

Paradigm?

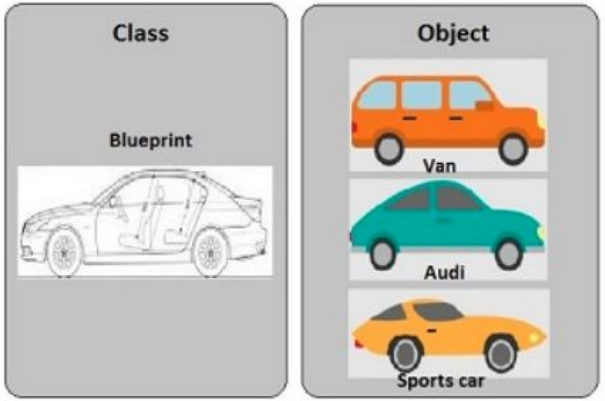


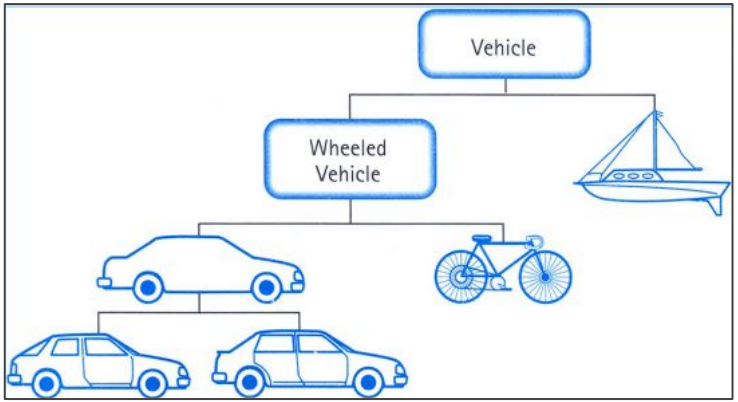
How about JavaScript?
Functional/OOP/Multi Paradigm?

BINAR ACADEMY

2. OOP Concept

Why use OOP?





BINAR ACADEMY

Oops wait...
take a look...

```
let today = {
  year: 2017
  month: 12,
  day: 24,
};

let tomorrow = (today) => {
  let year = today.year
  let month = today.month
  let day = today.day + 1
  return {year: year, month: month,
day: day}
};

let dayAfterTomorrow = (today) {
  day: tomorrow.day
  return day
};
```

```
class SimpleDate {
  constructor(year, month, day) {
    this._year = year;
    this._month = month;
    this._day = day;
  }

  addDays(nDays) {
    this._day += nDays
  }

  getDay() {
    return this._day;
  }
}
```

Public vs Private

Public can make variable/function available to be accessed from anywhere (other classes and instances of the object),

while Private cannot.

Douglas Crockford (leaders in the JavaScript community) said:

> *Do not use _ (underbar) as the first character of a name.*
>
> *It is sometimes used to indicate privacy, but it does not actually provide privacy.*
>
> *If privacy is important, use the forms that provide private members.*
>
> *Avoid conventions that demonstrate a lack of competence.*

For more details: article

Getter & Setter

The **get** syntax binds an object property to a function that will be called when that property is looked up.

The **set** syntax binds an object property to a function to be called when there is an attempt to set that property.

Example

```
class Person {
  constructor(value) {
    this._name = value
  }

  get name() {
    return this._name
  }

  set name(value) {
    this._name = value
  }
}
```

```
var c = new Person('John')

//get
console.log(c.name)
  -> 'John'

//set
c.name = 'Doe'

//get
console.log(c.name)
  -> 'Doe'
```

BINAR ACADEMY

Coverage

- → Abstraction
- → Encapsulation
- → Inheritance
- → Polymorphism

3. Abstraction

Hiding internal details and show only relevant data to user.

Abstraction manages complexity of a system by hiding internal details and composing it in several smaller systems.

BINAR ACADEMY

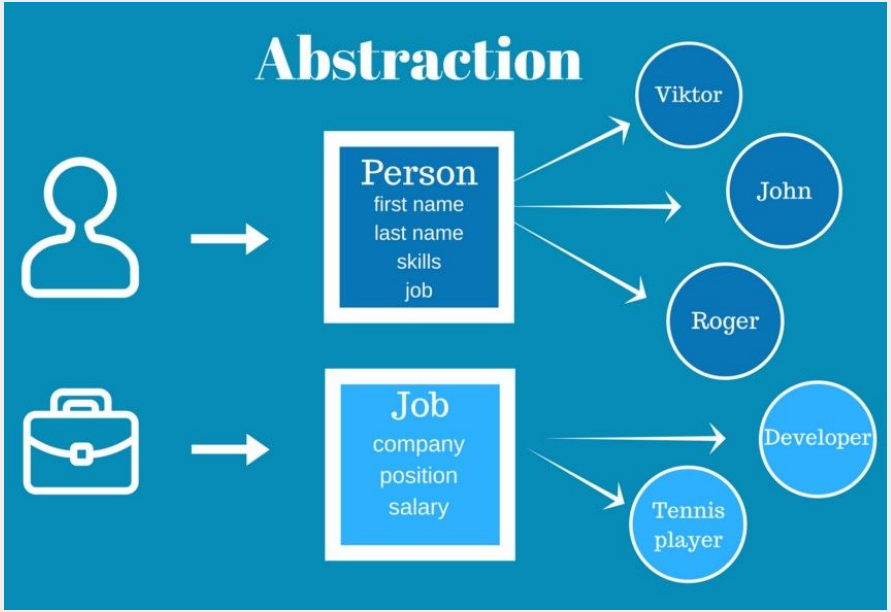
Let's consider an example.

We need a list of people in scope of our application, and we need to know their

first and last name, skills, job, and salary,

but in the same time we don't need

the age, height, weight.



Example: Abstraction

```
class Person {
   constructor({firstName, lastName, job}) {
       this.firstName = firstName;
       this.lastName = lastName;
       this.job = job;
       this.skills = [];
       Person._amount = Person._amount || 0;
       Person._amount++;
   }


   static get amount() {
       return Person._amount;
   }

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


   set fullName(fN) {
       if (/[A-Za-z]\s[A-Za-z]/.test(fN)) {
           [this.firstName, this.lastName] = fN.split(' ');
       } else {
           throw Error('Bad fullname');
       }
   }


   learn(skill) {
       this.skills.push(skill);
   }
}
```

```
class Job {
   constructor(company, position, salary) {
       this.company = company;
       this.position = position;
       this.salary = salary;
   }
}


const john = new Person({
   firstName: 'John',
   lastName: 'Doe',
   job: new Job('Youtube', 'developer', 200000)
});


const roger = new Person({
   firstName: 'Roger',
   lastName: 'Federer',
   job: new Job('ATP', 'tennis', 1000000)
});


john.fullName = 'Mike Smith';
john.learn('es6');
roger.learn('programming');
john.learn('es7');
```

4. Encapsulation

Process to hiding irrelevant data from user.

Binds the code and the data together and keeps them safe from outside interference.

BINAR ACADEMY

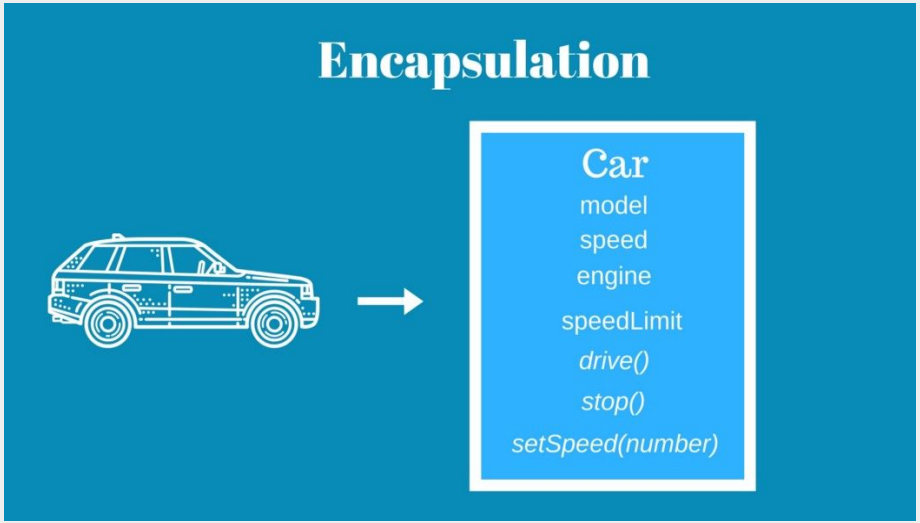
Let's consider an example.

In this case, we will need a car properties:

model name, current speed, max speed, and boolean engine prop that will be responsible for a state if a car is turned on or off.

Encapsulation principle means that we should add to the same class behavioral methods (drive, stop, etc.).

Those may be used in our application and also to provide restricted access to changes in class instance's state.



```
// Encapsulation example

class Car {
  constructor(model, price) {
    this._model = model
    this._price = price
  }


  get price() {
    return this._price
  }
}

var c = new Car('Tesla', 20000)
console.log(c.price)
```

5. Inheritance

Car

The process of acquiring and extending characteristics for the child from its parent. One object inherit the characteristics of another object.

Example:
honda is a car,
toyota is a car can add more methods

BINAR ACADEMY

Inheritance

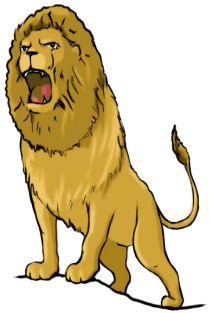
```
// Inheritance example

class Car {
  constructor(model, price) {
    this._model = model
    this._price = price
  }

  get price() {
    return this._price
  }

  brush(color) {
    console.log(`${this._model} color is ${color}`);
  }
}
```

```
class Toyota extends Car {
     constructor(model, price, cc) {
          super(model, price);
          this._cc = cc;
     }


  start() {
    console.log(`The ${this._model}
engine is started`)
  }
}

var t = new Toyota('Yaris', 2000, 1500)
t.brush('red')
t.start()
```

6. Polymorphism

speak:
meow

speak:
roar!



The process of using
same method name by multiple classes and
redefines methods for the derived classes.

BINAR ACADEMY

Polymorphism (Overriding)

```
class Car {
  constructor(model, price) {
    this._model = model;
    this._price = price;
  }

  get model() {
    return this._model;
  }

  get price() {
    return this._price;
  }

  brush(color) {

console.log(`${this._model}
color is ${color}`);
  }
}
```

```
class Toyota extends Car {
  constructor(model, price, cc) {
    super(model, price)
    this._cc = cc
  }
  brush(color) {
    super.brush(color)
     console.log(`Prepare to test engine ${this._cc} cc!`);
  }
  start() {
    console.log(`The ${super._model} engine is started`)
  }
}

var t = new Toyota('Yaris', 2000, 1500)
t.brush('red')
t.start()

> "Yaris color is red"
> "Prepare to test engine 1500 cc!"
> "The Yaris engine is started"
```

Benefits:

- ➔ Extensibility
- ➔ Reusability
- ➔ Eliminate redundancy

Exercise

Buatlah sebuah Class Student, yang memiliki atribut berikut:

- Name,
- Age,
- Date of Birth,
- Gender
- Student ID (bisa berupa angka atau teks), dan
- Hobbies (bisa menampung lebih dari 1 hobi).

Class tersebut juga bisa memanggil fungsi dengan proses sebagai berikut:

- SetName: mengubah nama student dengan mengirimkan satu parameter ke dalam fungsi berupa teks
- SetAge: mengubah umur student dengan mengirimkan satu parameter ke dalam fungsi berupa angka
- SetDateOfBirth: mengubah tanggal lahir student dengan mengirimkan satu parameter ke dalam fungsi berupa teks
- SetGender: mengubah gender student dengan mengirimkan satu parameter ke dalam fungsi berupa teks, dan hanya bisa menerima nilai `Male` atau `Female`
- addHobby: menambah hobi dengan mengirimkan satu parameter ke dalam fungsi berupa teks
- removeHobby: menghapus list hobi yang ada dengan mengirimkan satu parameter berupa teks, yang merupakan hobi apa yang akan dihapus
- getData: menampilkan seluruh data atribut murid