OOP

Programming

procedural Programming and Object oriented programming

Procedural Programming

- [ ] Code is not reusable.(same code repeat and repeat)
- [ ] Large code base is hard to manage
- [ ] Difficult to tracing & fixing bugs
- [ ] Data is exposed to whole program (all person data are show for all it's huge problem us it's a big security issue).
- [ ] Operation's priority is higher then data (data is everything but procedural operation priority is higher).

What is OOP?

OOP Programming is all about object.

▼ Class

Class is a template/ blueprint to create multiple object. class have property/ attribute and method

Method

- [ ] Private method used in this to do anything.
- [ ] public method access to create object.

Class only show data type/ shape of data(**Objects that will contain real data)** class pass data use Constructor.

template is equal to class. Real data is equal to object.

Use Class template to create object(object have access methods(not method only access) and property) and same data are pass two time it's create two different object.

▼ Object

Anything that needs multiple Attributes or multiple Primitives to define.

- [ ] Combination of Noun, Adjective (property name) & verb(method)
- [ ] A Capsule That can Encapsulate Data & Operations
- [ ] Has some private(class theke access korte hobe) and public properties (object theke access kora jabe)
- [ ] Has some Functions Which is Called Methods.
- [ ] Is a Custom Data Type (class use kore object create korle class ta hoye jay akta custom data type).
- [ ] An isolated environment for Properties & Methods.(object ke baire theke kisui bole dite hobe na sop kisu o nijei jane kivabe ki korte hobe ar moddei sop logic deya ace baire theke kono kisui dite hobe na).

Finding Object, It's Properties and Methods are the Main Mechanism of Learning Object oriented Programming

Things That Are important In OOP

> OOP Has Four Main Pillar

▼ Abstraction

Abstraction is the process of hiding complex implementation details and providing a simpler interface for the user. This simplification allows the user to interact with the code more easily, without needing to know all the underlying details of how it works. Abstraction can be achieved through the use of abstract classes, interfaces, and inheritance.

একটি ক্লাস তার নিজের মধ্যে সপ কিছু হাইড করে রাখবে এবং অন্য কাকে কিছু বলবে না। তারা শুধু কল করবে এবং কাজ হবে।

we have the controller to control this Mp3 Player But We Don't know How the Device is working That is called Abstraction

we don't know how to create or generate today date in Date() Class. Implementation Details hide kora. it's call Abstraction.

▼ Encapsulation

Encapsulation is the process of hiding an object's internal state from the outside world and restricting access to it through public methods. This allows for better control over how data is accessed and modified, ensuring that it is only done in a safe and controlled manner. Encapsulation also helps to prevent external code from interfering with the internal workings of an object, improving overall code stability and reliability.

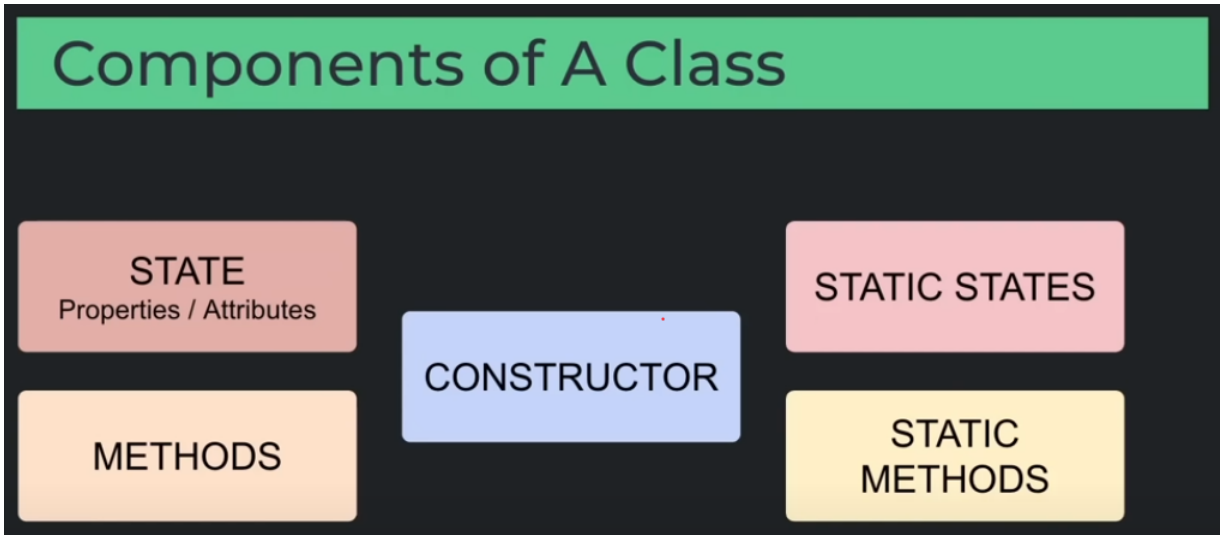
Encapsulation means that each object in your code should control it’s own state. State is the current “snapshot” of your objcet.

Abstraction এবং এনক্যাপসুলেশন কচ্ছ কচ্ছ একটি টার্ম। Abstraction সম্পূর্ণরূপে করার জন্য এনক্যাপসুলেশন প্রয়োজন। যদি আমরা ইমপ্লিমেন্টেশন বিশয়গুলি লুকিয়ে রাখতে চাই সপ কিছুকে আমাদের একটি জায়গায় encapsulated করতে হবে। এনক্যাপসুলেশন একটি উপায় যে উপায়ে আমরা Abstraction করতে পারি।

```
class Person {
  private String name;
  public static int key;

  public Person (String name){
    this.name =name;
  }
  public String getName () {
    return this.name;
  }

  public static Person create(String name) {
    return new Person(name);
  }
}
```



▼ Inheritance

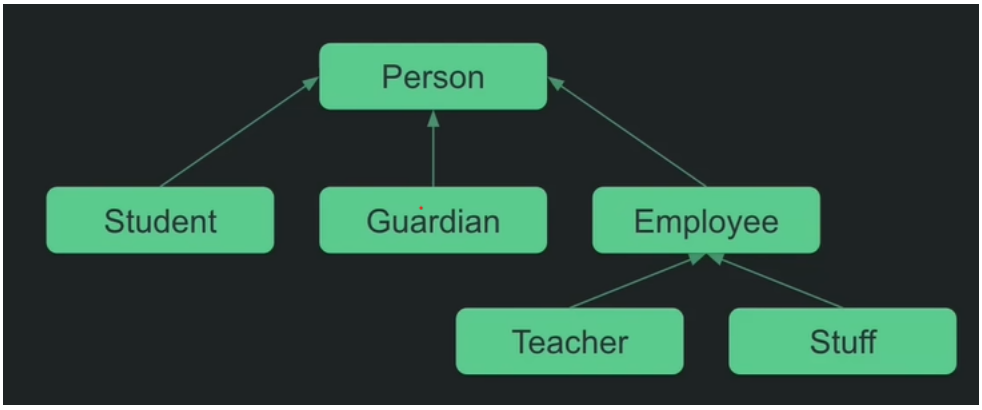
- Inheritance allows a class to inherit properties and methods from a parent class, reducing code duplication and promoting code reuse.
- The ability of creating a new class from an existing class. Inheritance allows a class to acquire the properties and behavior of another class.

কোড নকল করতে হবে না এবং কোড পুনরাবৃত্তি কম করতে হবে।

There are two type of Relation in OOP :

> 💡 IS A RELATION → Inheritance

ইনহেরিটেন্স মানেই একটি সম্পর্ক।

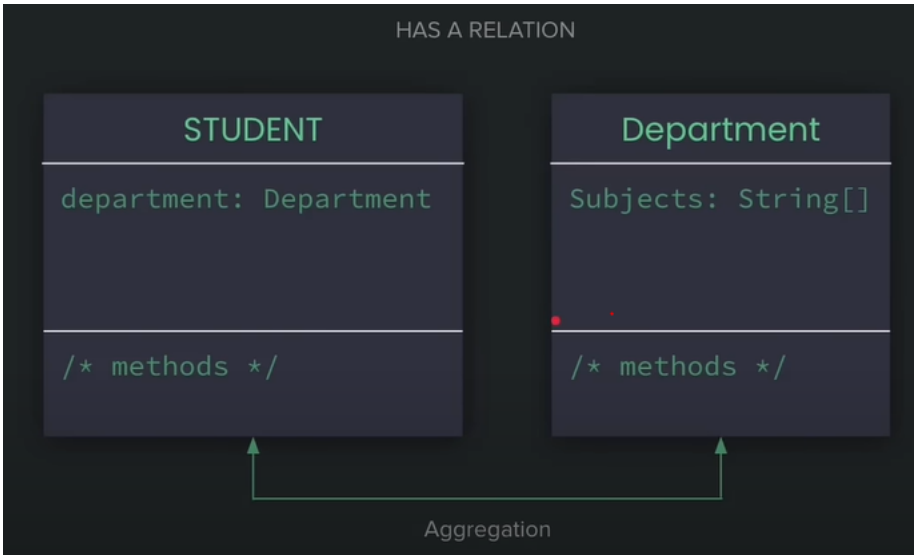


Ex : Student is a Person / Employee is a Person.

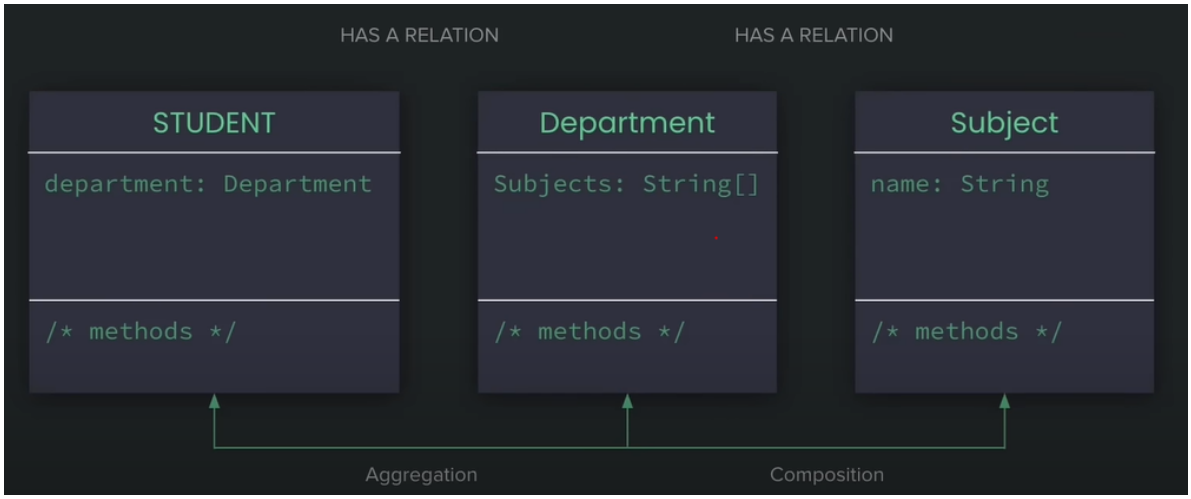
> 💡 HAS A RELATION → Composition & AGGREGATION

আমার কাছে যখন আছে তখন ‘has a relationship’.

যদি এমন হয় যে একটা অবজেক্ট আরেকটা অবজেক্টের উপর নির্ভর করে না, তাহলে তা হয় "অ্যাগ্রিগেশন"।



কোন কারণে যদি একটা অবজেক্ট মুছে যায়, তবে সে সঙ্গে যুক্ত অন্য অবজেক্টগুলোও মুছে যাবে এটি হচ্ছে "কমপোজিশন"।



উদাহরণস্বরূপ: যদি কোনো কারণে ডিপার্টমেন্ট মুছে যায়, তবে সাথে যুক্ত সাবজেক্টগুলো ও মুছে যাবে এটি কমপোজিশন। এবং যদি সাবজেক্টটি অন্য কারো সাথে যুক্ত করা যায়, তবে তা হয় অ্যাগ্রিগেশন।

যেখানে চাইল্ড স্বয়ংক্রিয়ভাবে নিজেই থাকতে পারবে, তারও বিষয়টি হচ্ছে Aggregation.

যেখানে চাইল্ড স্বয়ংক্রিয়ভাবে নিজেই থাকতে পারবে না, তারও বিষয়টি হচ্ছে কমপোজিশন।

Upcast kivabe kore ?

```
class student extends Person {
  constructor(name, email, subjects, fee){
    super(name, email); // parent class ar constructor ke call korbe tar parameter diye dite hobe.

    this._subjects = subjects;
    this.fee = fee;
  }

  super.print() // super classcall korar maddome amra super calss ar sop kisu call korte parbo.
```

▼ Polymorphism

Polymorphism is the ability of an object to take on many forms. This allows different objects to be treated in a similar way, even if they have different properties or behaviors. Polymorphism is achieved through the use of interfaces, abstract classes, and method overloading/overriding.

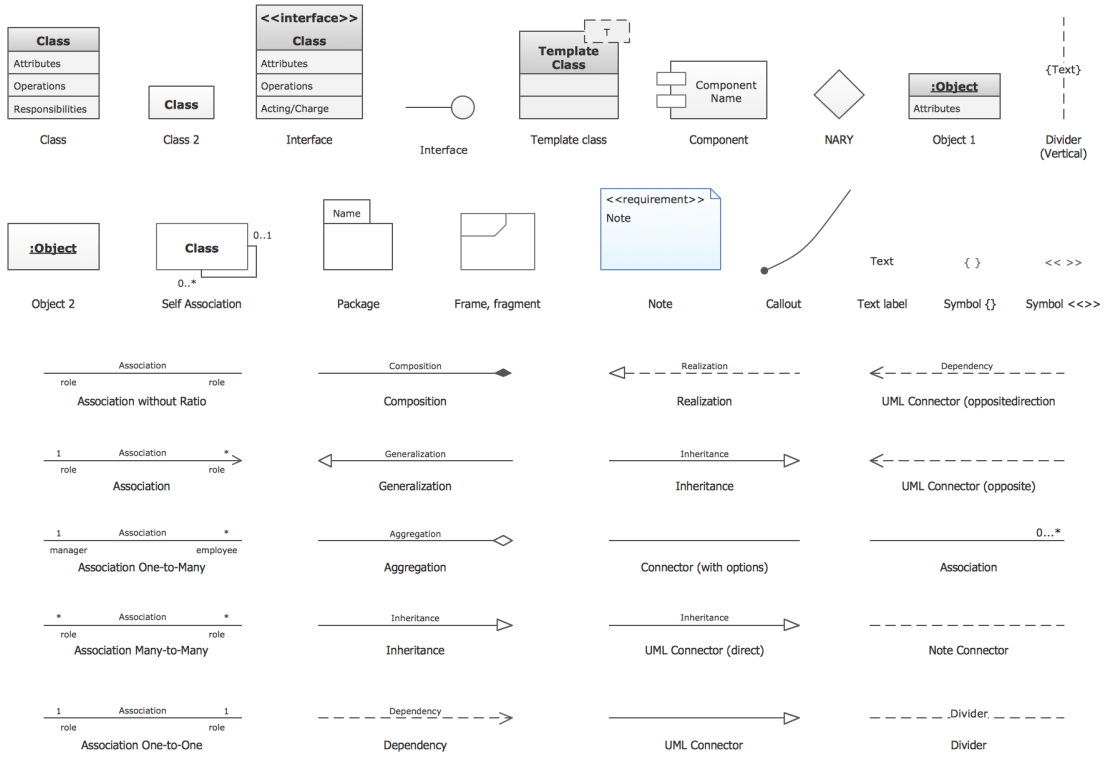
একটা ক্লাস যখন আমরা ইনহেরিট করি তখন বিভিন্ন জায়গায় বিভিন্ন রকম ভাবে তার বয়সিত তুলে আনতে পারে। ইনহেরিটেন্স এবং সঙ্গে সম্পূর্ণরূপে সম্পর্কিত পলিমরফিজম। একটি ক্লাস থেকে অনেকগুলি অবজেক্ট তৈরি করতে পারি যদি তাদের প্যারেন্ট ক্লাস এক হয় তবে একটা বইসিত থাকবে যদি বিন্যাস চেঞ্জ হয় তবে বিনো বইসিস্ট হবে।

> 💡 POLYMORPHISM

- Compile Time →
  - Constructor Overloading

    আধুনিক ভাষা দেখা যায় না, যেমন Java, C#, C++, কনস্ট্রাক্টর ওভারলোডিং নিয়ে একটি বিষয় আছে। আমি একই কনস্ট্রাক্টরে বিভিন্ন ধরনের মান প্রদান করার মাধ্যমে বিভিন্ন কনস্ট্রাক্টর কল করতে পারি, তাতে আমাকে কন্ডিশন চেক করতে হবে না, কনস্ট্রাক্টর ওভারলোডিং এই কাজটি করবে।
  - Operator Overloading
    - C++ এ অপারেটর ওভারলোডিং রয়েছে। আমরা আমাদের মতো করে কাস্টম অপারেটর তৈরি করতে পারি এখানে। একটি ক্লাসে যখন নাম্বার দিয়ে অপারেটর ব্যবহার করা হয়, তখন সে যোগ করে। আবার স্ট্রিং ব্যবহার করা হলে সে কানক্যাটেনেট করে।
  - Function Overloading
    - একটি একই ফাংশন বিভিন্ন ইনপুটের জন্য বিভিন্ন আউটপুট উত্পাদন করবে।
  - Function Overriding
    - প্যারেন্ট ক্লাসে একটি ফাংশন সংজ্ঞায়িত আছে, এটি চাইল্ড ক্লাসে আমি অন্য ভাবে সংজ্ঞায়িত করবো বা ব্যবহার করবো, এটি আসল বৈধতা হারিয়ে যাবে না, এটি হচ্ছে ফাংশন অভাররাইডিং (Function Overriding)।
- Run Time ⇒ Virtual Function
  - এই ফাংশনগুলো রানটাইমে সময়ে আকৃতি (shape) পরিবর্তন করতে পারে এবং ফাংশনটির পদ্ধতি (way) পরিবর্তন করতে পারে।

OOP Language



Access Modifier

JavaScript has not support access modifier.

JavaScript overloading possible na override possible.

JavaScript follow community driven solution.

Community driven solution bolte amra akta convention follow korte pari ate compiler bujbe na je private / public but developer bujte parbe convention diye. as like <_email> arokom underscore deya gulote tara access kore na.

jehetu amader javascript compiler ar kase public/private/protected kono modifier nei tai _ (underscore) deya variable ke amra getter and setter diye access korbo.

```
class Person {
  constructor(name, email) {
    this._name = name;
    this._email = email;
    console.log(this); //akhane this bolte ai class ar jotogulo object ace tader bujay.
                       // this == p1 abr this p2 jokhon je object niye kaj hobe seta.
  }
  changeNamr(name){
    this._name = name;
  }
  sendEmail(msg){
    console.log('To', this._email); // this.email set hobe jokhon sei object theke call hobe tar email.
    console.log('Sending Email');
    console.log('MSG', this._sanitizeMsg(msg));
  }
  _sanitizeMsg(msg){
    return msg.trim().toLowerCase();
  }
```

```
}

const p1 = new Person ('sakil', 'sakil@gmail.com');
const p2 = new Person ('sakil1', 'sakil1@gmail.com');

console.log(p1);
console.log(p2);

p2.sendEmail('   hello');
p1.sendEmail('hi     ');
```

_sanitizeMsg(msg) ata bairer karo janar dorkar nai ata private method.

toString Override

polymorphism ar 2 ta kaj method overload kora ar override kora.

jokhon amra console.log(p1) korci tokhon e

> 💡 Person { name: 'sakil', email: 'sakil@gmail.com' }

output asce. ata kotha theke asce seta holo toString.

toString method ke amra amader moto customize korte pari. toString tokhon call hoy jokhon string concat korar dorkar hoy / jokhon class a dynamic vabe dorkar hobe string value.

> 💡 console.log(p1 + “ “) // akhon toString method call hobe and ar modde ja return kora ace tai back korbe. akhon jodi toString method override kora na thakto tahole output ascto [object object]

```
print() {
    console.log(this + ""); // Person { name: 'sakil', email: 'sakil@gmail.com' } ai output dibe.
  }
toString(){
    return `Name : ${this._name}, Email : ${this._email}`;
  }

const p1 = new Person ('sakil', 'sakil@gmail.com');
p1.print();

console.log(p1 + “ “) // same output e dibe.
```

Getter And Setter

```
class Person {
  constructor(name, email) {
    this.name = name;
    this.email = email;
  }

  get name() { // ata akta function ami chaile p1.name = "dsf" kisui bosiye dite parbo na.
    return this._name; // function call na korei private data get korar system paowa gelo.
  }
  set name(value){
    this._name = value;
  }
  print() {
    console.log(this);
  }
}

const p1 = new Person ('sakil', 'sakil@gmail.com');
const p2 = new Person ('sakil1', 'sakil1@gmail.com');
```

```
p1.name = 'New Name'; // jodi get function na likhe aita lekha hoto tahole object a new akta property jukto hoye jeto. amader kaj chilo exi
// jodi set function lekhar pore ata change kora hoy tahole auto name ar value set function ar maddome change hoye jabe.
console.log(p1.name); // get ar name ke name() avabe call kora lagbe na sudu object property moto access korlei hobe as like p1.name but be
p1.print()
```

Static Method

jekono akta **object** kon **class** ar ta janar jonno **instanceof** use kora hoy.

💡 console.log(p1 instanceof Person); // true

jodi ata person ar create kora object na hoy tahole false dibe.

Static property/method holo object instance create na kore directly class theke access korar.

💡 static className = “Person”;

Person.className;

💡 static is(age){
return age ≥ 18;
}
Perosn.isValid(20);

Private Property using Symbol

💡 class ar baire declare korte hobe
const *name = Symbol('name');*
class ar modde access korte hobe [] diye.
this[_name] = name; // dot notation ar poriborte array notation.

OOP Case Studies

▼ Client Requirements:

Build a University Management System where admin can manage students, guardians, teachers, staff, departments, subjects, exams and Accounts.

Admin can update, create, delete and manage all the resources.

▼ Breakdown The Requirements:

Object:

1. Student
2. Guardian
3. Teacher
4. Staff
5. Department

6. Subject
7. Exam
8. Account

Student:

- id
- name
- guardian
- contact
- blood
- account
- exams
- department
- subjects

Guardian:

- id
- name
- contact
- blood
- profession
- income

Contact:

- id
- email
- phone
- alternative phone
- address

Address:

- id
- road no
- city
- region
- country
- postal code

Teacher:

- id
- name
- contact
- blood
- department
- subject
- salary

Staff:

- id
- name
- contact
- blood
- department
- title
- salary

Department:

- id
- name
- subjects
- teachers
- dean (Teacher)

Subject:

- id
- name
- credit
- department

Exam:

- id
- name
- pass mark
- duration
- subject
- student

Account:

- id
- type
- amount
- time

▼ Class Property Method Object And Constructor

```
class Person {
  constructor(name, email) {
    this.name = name;
    this.email = email;
    console.log(this); //akhane this bolte ai class ar jotogulo object ace tader bujay.
                      // this == p1 abr this p2 jokhon je object niye kaj hobe seta.
  }
  changeNamr(name){
    this.name = name;
  }
  sendEmail(msg){
    console.log('To', this.email); // this.email set hobe jokhon sei object theke call hobe tar email.
    console.log('Sending Email');
    console.log('MSG', msg);
  }
}

const p1 = new Person ('sakil', 'sakil@gmail.com');
const p2 = new Person ('sakil1', 'sakil1@gmail.com');

console.log(p1);
console.log(p2);

p2.sendEmail('hello');
p1.sendEmail('hi');
```