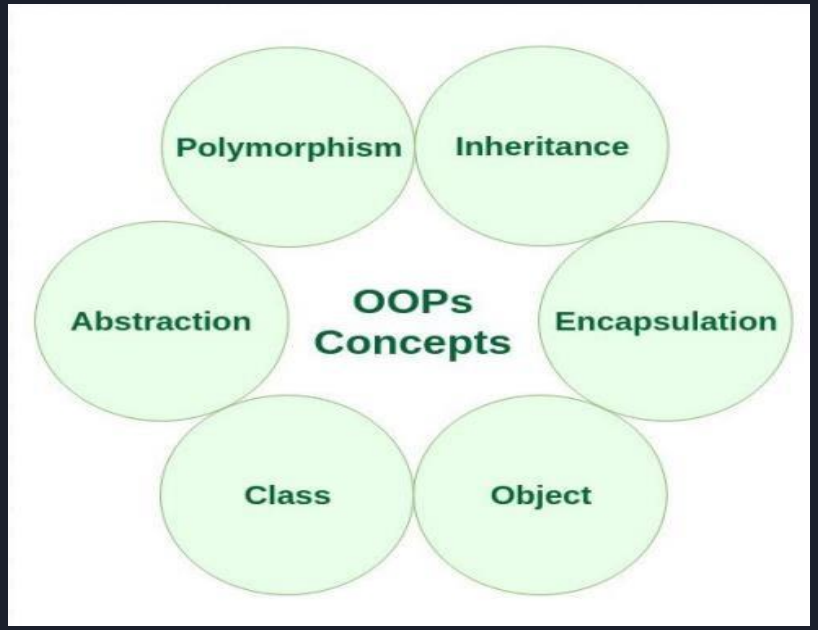
Small intro about classic OOP

Object Oriented Programming

Abdelrahman Shaheen

Content



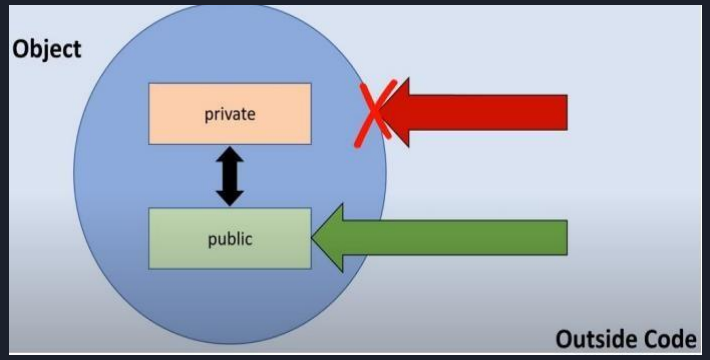
Note : Close correspondence between real-world objects and OOP classes.

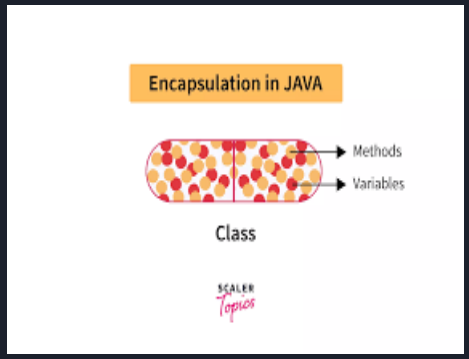
Classes and Objects

- Blueprint ?

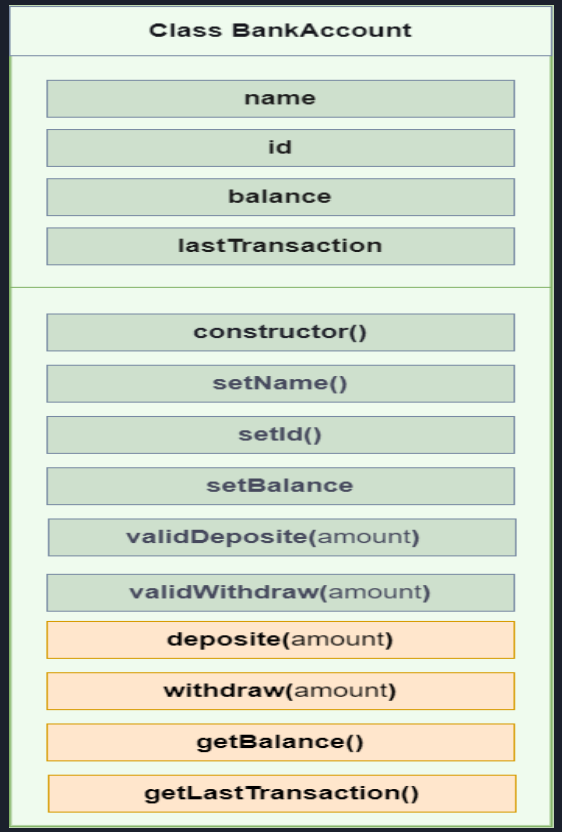
Encapsultaion

- Encapsulation ? [Variables +Functions] +Data Hiding/Security
- Data Hiding ? Access modifiers [public , private and protected]
- Security ? Validation of the data.
- No data hiding → Weak encapsulation +More complexity
- With data hiding → Strong/Good encapsulation +Complexity is hidden.
- Why all of this ? Prevents external code from accidentally manipulating internal properties/state





Applying Encapsulation on BankAccount Example

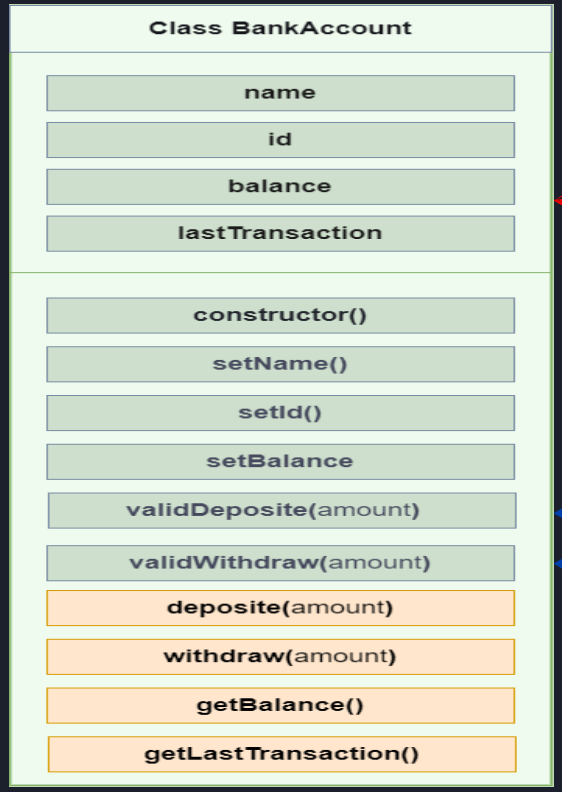


- Your task is : to define which function is public and

which one is private (Apply the data hiding and Encap)

Main Interfaces/Functions

Solution



Private data

Private functions

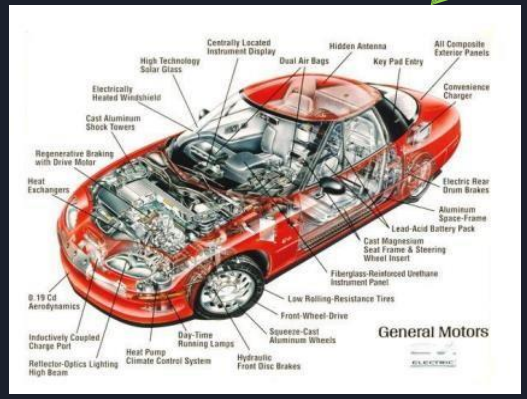
- Setters , deposite ,withdraw ,getBalance and getLastTransactions are called public interfaces (APIs)

Abstraction

- Abstraction is about hiding unwanted details while showing most essential in a given context .
- Real life example : (Cars , internet , computers , ……etc)

Real Car

Abstracted Car



- By nature , encapsulation leads to abstraction , because encapsulation hides the object details ,

  Thus **encapsulation is required for abstraction.**

Abstraction

Examples in the code :

- BankAccount
- String class or any object in javascript.

Inheritance

- Inheritance: Making all properties and methods of a certain class available to a child class, forming a hierarchical relationship between classes. This allows us to reuse common logic and to model real-world relationships.

Parent/Base/Super Class

```
public class Person{

    protected String name = "abdelrahman";
    protected String email = "shaheen@gmail.com";


    protected boolean isValidEmailFormat(){
        //validation logic
    }

}
```

Inherited

Inheritance

Child class extends parent class

Inherited

Child/Derived/Sub Class

```
public class Student extends Person{

    private double gpa;

    String name = "abdelrahman";
    String email = "shaheen@gmail.com";

    public void printGrades(){
        //printing logic
    }
    public void setGpa(){
        //some logic
    }

    protected boolean isValidEmailFormat(){
        //validation logic
    }
}
```

Inheritance

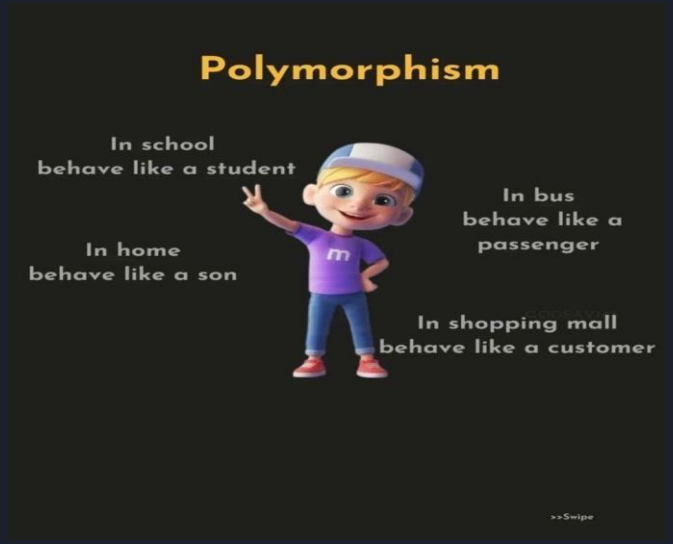
- The main idea of inheritance is to reuse the code ,So no duplication code or no efforts.
- But there is an important condition here : There is must be a relationship between base/sub class.
- What kind of relationship ? **IS-A relationship.**
- Can human inherit from animal because they've similar functions ? Like eating , sleeping☺
- So before applying the inheritance , Check the relationship.

Is-A relationship

- Student is-a person. Teacher is-a **person**
    - So some common variables/functions + some unique variables/functions.
- Circle is-a shape , Rectangle is-a shape. Triangle is-a **shape**.
- Software Engineer is-an employee , Manager is-an employee , Office Boy is-an **employee**.
- Apple is-a fruit. Orange is-a fruit. Watermelon is-a **fruit**.
- (Bicycle / Motor vehicle / Railed vehicle) is-a **vehicle**

Polymorphism

The ability of an object to identify as more than one type.



Object oriented programming in Js

- There are 3 ways to implement oop in js
  - Constructor functions
  - ES6 Classes
  - Object.create()

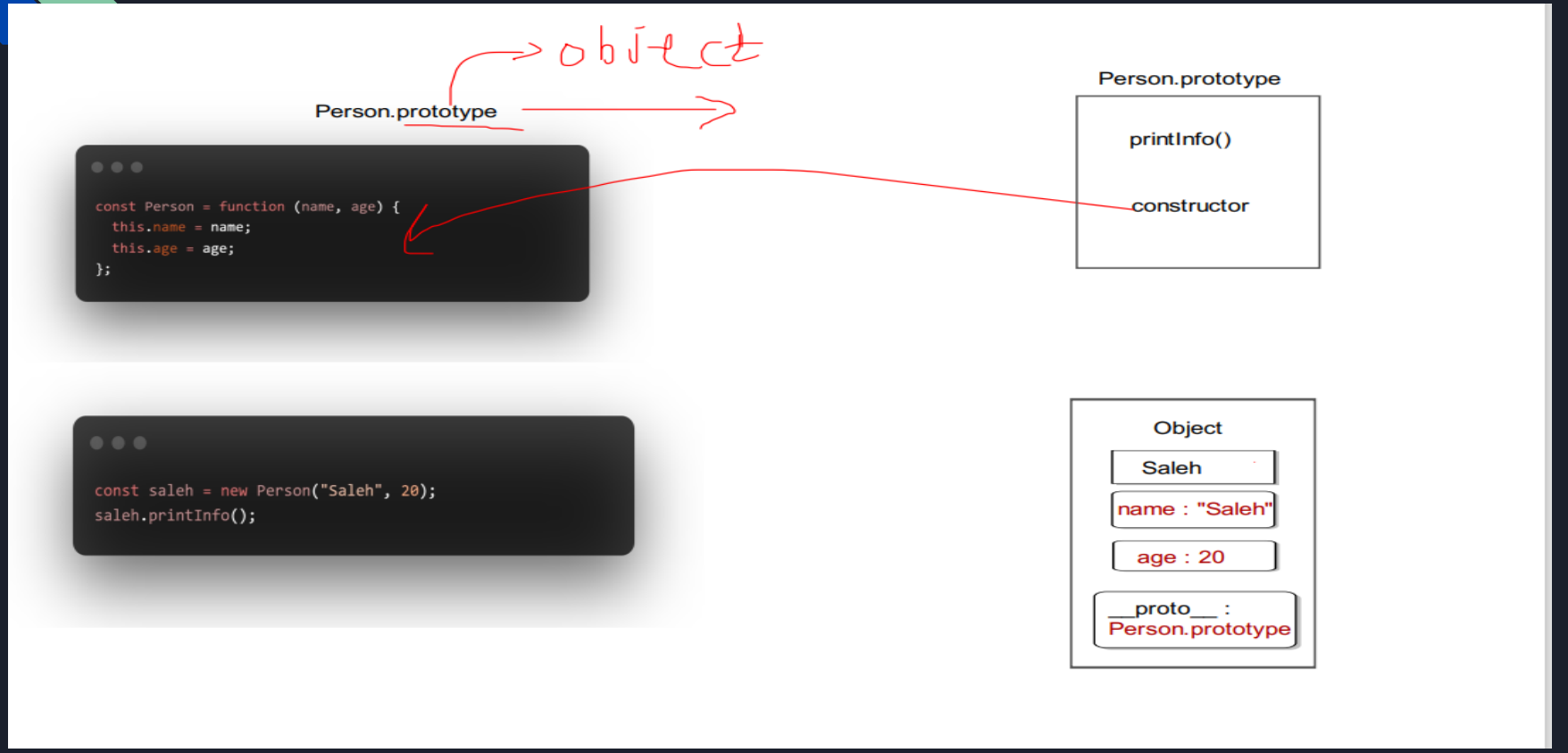
Constructor function

- We can use constructor functions, to build an object using a function.
- a constructor function is actually a completely normal function. The only difference between a regular function, and a function that we call constructor function, is that we call a constructor function with the ***new*** operator.
  NOTE : we write it as function declaration or expression ,cuz arrow ,doesn't have its own this keyword
- Code 1
- What actually happens behind the scene , When we call a function with "new" operator ?

1. New { } is created.
2. Function is called , and in the EC , this → { }
3. { } linked to a prototype.
4. Function automatically return { }

Constructor function

- Implementing functions inside a constructor function !!!!!!!!!!!!!!!!!!!!!!!!
- Imagine you will create a hundred or thousand of objects from this constructor !
- Another way to create a method ? ***Prototypes*** ?
- See code (2)


object
Person.prototype
const Person = function (name, age) {
this.name = name;
this.age = age;
};
Person.prototype
printInfo()
constructor
const saleh = new Person("Saleh", 20);
saleh.printInfo();
Object
Saleh
name : "Saleh"
age : 20
__proto__ :
Person.prototype

Prototypes

- 1. Every function in JS will automatically have an property called prototype, which includes constructor function.
  Note1: Function in JS is essentially an object, so that it can have property
  Note2: Constructor functions have NO difference with other regular functions until we invoke a constructor function with new keyword

- 2. prototype is a property on function which contains all the useful properties that will be inherited by it's instance

- 3. Every object created by a constructor function will be able to get access to all the methods and properties that set on the prototype of the constructor function

- 4. By this way, we don't need to set the method on every object. Instead, we only set the method once on the prototype of constructor function. Then all the object created by this constructor function will have access to that method

__proto__

- 1. Every object in JS have a property called __proto__
- 2. The __proto__ property of object is essentially the prototype property of the constructor function that create the object
- 3. In other words, __proto__ of object is as same as prototype of function which creates that object
- 4.What creates the __proto__ ? ANS: step 3 when we use ***new*** operator.
- 5. See code (3)

__proto__

- Explination of code (3) : what happens inside the engine ?

- When we can call arr.at()
  The JS egine does something like this
- 1. Now I'm on the arr object(array). Is there a method called at() on arr object?
  No.
  It's fine, I just continue to find the at through it's __proto__
  JS engine go to prototype of Array function through __proto__.....

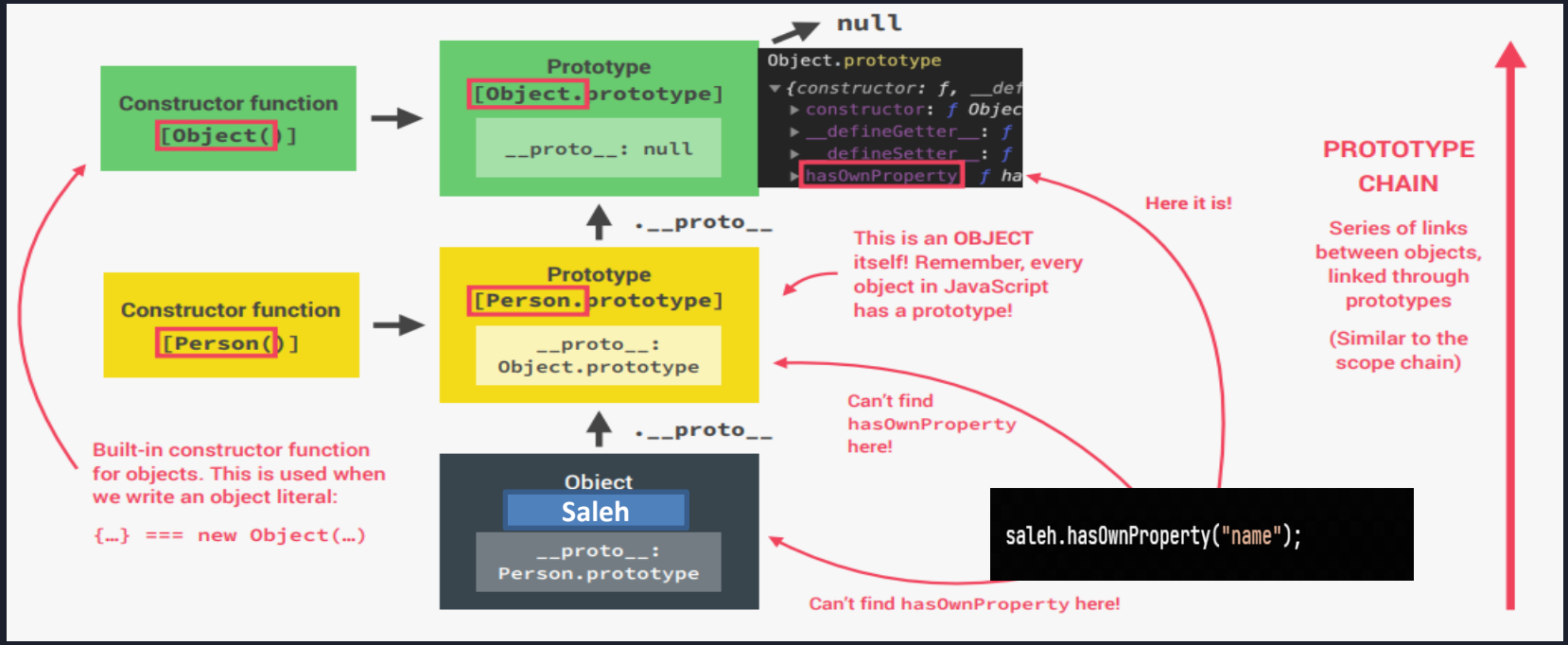
- 2. Now I'm at the prototype of Array function . Is there a method called at() on this object?
  Yes.
  Great, now I'm gonna use it.

➢ Look here : array

- This process called ***prototype chain***

Prototype Chain

- Chain : object -> object.__proto__ -> the proto of proto (object.prototype)



Prototype Chain

- Code 4
- **Object.prototype** (top of the prototype chain).

ES6 Classes

- Classes in JavaScript do not work like traditional classes in other languages like Java or C++. So instead, classes in JavaScript are just syntactic sugar over constructor functions.
- It abstracts all things of constructor functions.
- Classes are special kinds of functions.
- You can write it in two ways :
  - Class declaration
  - Class expression
- See code 5

ES6 Classes

- IMPORTANT NOTES:
  - Methods inside a class , will be added to .prototype property.
  - Classes are not hoisted.
  - Classes are first class citizens.
  - Classes are executed in strict mode.

Inheritance between classes