Object Oriented Programming (OOP)

Presenter: Edoga Chukwudi

What is OOP

Object-oriented programming (OOP) is a programming language model organized around objects rather than "actions" and data rather than logic. Historically, a program has been viewed as a logical procedure that takes input data, processes it, and produces output data.

What is OOP

Object-oriented programming takes the view that what we really care about are the objects we want to manipulate rather than the logic required to manipulate them. Examples of objects range from human beings (described by name, address, and so forth) to buildings and floors

Why learn OOP

You have heard about Object Oriented Programming before (OOP) but you are not too sure what it is and if you really need to learn a bit more about it. You have heard that if you want to use Java, javascript, create andriod apps you will need to learn OOP and you are now wondering if you should really start reading about it. Well good timing because that is what I am about to teach.

Why learn OOP

OOP forms the bedrock for programming at the enterprise level all useful applications are built with OOP concepts as the brick and mortar

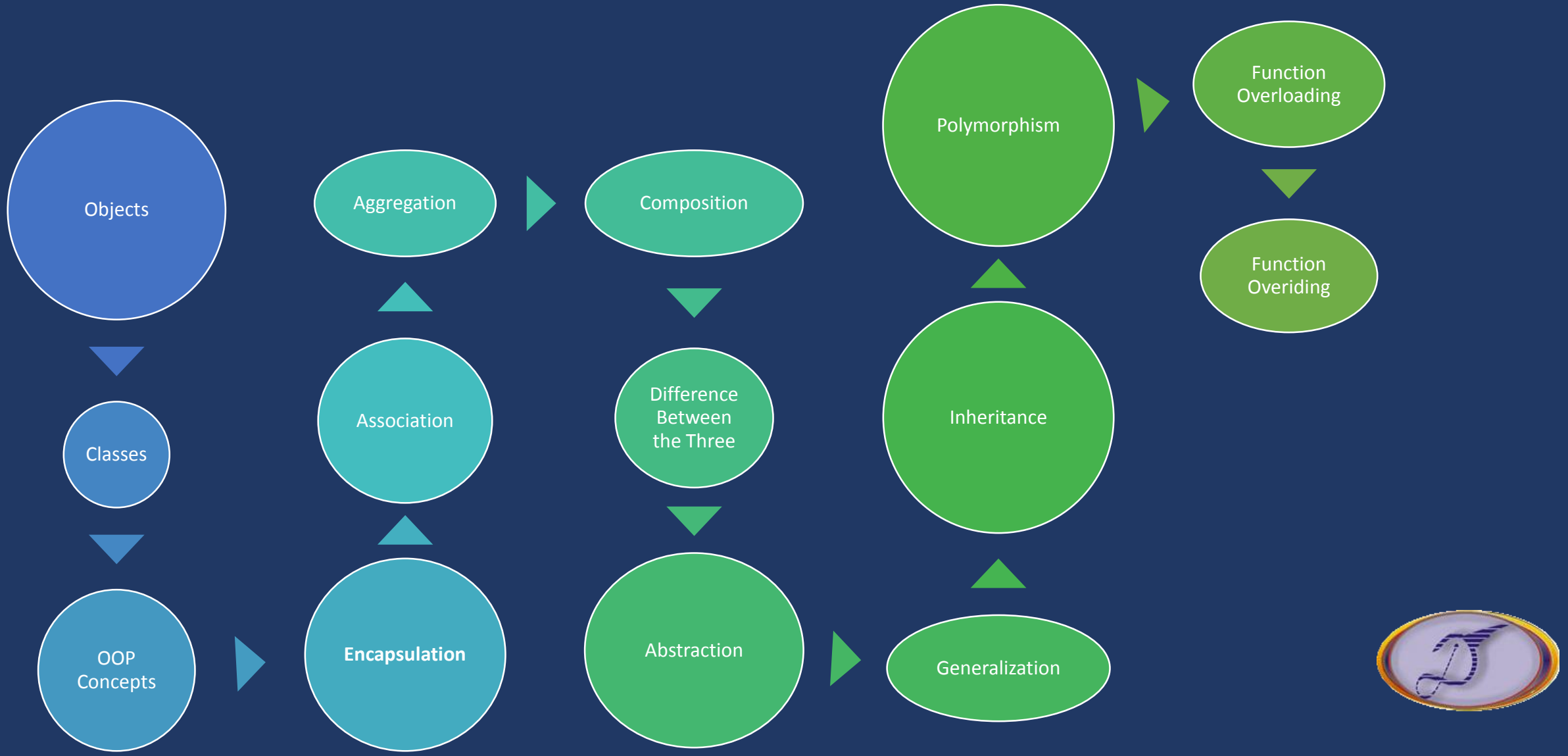
OOP is all around you

It is always scary for beginners to understand OOP concepts but you don’t need to be scared for OOP has been around you and you have used a bit of it.

OOP is all around you

Document Object Model is a good example of OOP. You have used it, toggled it properties and called its methods. So object are all around you so it is time to use objects to program.

What we will learn



Digital Dreams

Class

A *class* is simply a representation of a type of *object*. It is the blueprint, or plan, or template, that describes the details of an *object*. A class is the blueprint from which the individual objects are created. *Class* is composed of three things: a name, attributes, and operations.

Class Example

Digital Dreams

Student
• age : int • name : string
+ Student() : void + DoLearn(object) : boolean << Property functions >> + Name(): string + Age() : int

Objects

- An object can be considered a "*thing*" that can perform a set of related activities. The set of activities that the object performs defines the object's behaviour. For example, the Hand (object) can grip something, or a *Student* (object) can give their name or address.

Objects VS Classes

- In pure *OOP* terms an object is an instance of a class. A class is like a mould for manufacturing new object. Take for example, a mould for making cakes is the class while the cake that was produced is the object

Classes in javascript

- Javascript treats classes as associated array i.e
- Student.id === Student[“id”];

- Student[“id”] is an example of associated array or what is called a named key

Class in Javascript

Class is defined in javascript using the function keyword

```
Function Student ()
{
    this.name = “okeke”;
    this.speed=20;
    this.getSpeed=function()
    {
            return this.speed;
    }
    this.talk=function(){ alert(this.name) ; }
}
```

Objects in Javascript

You can create a new object in javascript using the object factory. The object factory is defined with the ‘Object’ keyword.

Objects created in javascript has all properties public.

When we say public we mean that the member variables can be manipulated after the object has been instatianted.

Using ‘new Object()’


```
person = new Object() ;
person.name = "Tim Scarfe“;
 person.height = "6Ft" ;
person.run = function()
 {
    this.state = "running“;
     this.speed = "4ms"
}
```

Using Literal Notation

Object are enclosed in curly brackets with properties separated by comma and key-value pairs separated by colon.

```
var rectangle = {
    upperLeft : { x : 2, y : 2 },
    lowerRight : { x : 4, y : 4}
}
```

Object constructors

Contructors are used to instantiate a class i.e it is used to create a new object from the class.

```
var pupil = new Student();
pupil.talk();
```

OOP Concepts

- There are four main gods of *OOP* world and in software term, they are called four main Object Oriented Programming (*OOP*) Concepts.

1. Encapsulation
2. Abstraction
3. Inheritance
4. *Polymorphism*.

Encapsulation

The encapsulation is the inclusion-within a program object-of all the resources needed for the object to function, basically, the methods and the data. In *OOP* the encapsulation is mainly achieved by creating classes, the classes expose public methods and properties.

Encapsulation

A class is kind of a container or capsule or a cell, which encapsulate a set of methods, attribute and properties to provide its indented functionalities to other classes. In that sense, encapsulation also allows a class to change its internal implementation without hurting the overall functioning of the system. That idea of encapsulation is to hide how a class does its business, while allowing other classes to make requests of it.

Association

- Association is a (*a*) relationship between two classes. It allows one object instance to cause another to perform an action on its behalf. Association is the more general term that define the relationship between two classes, where as the aggregation and composition are relatively special.

Association Example


```
Function Student(Registrar)
{
    this.RegNo= Registrar.assignReg();
}
```

Aggregation

Aggregation is a *weak* type of **Association** with *partial* ownership. For an **Aggregation** relationship, we use the term *uses* to imply a *weak* *has-a* relationship.

Weak meaning the linked components of the aggregator may survive the aggregations life-cycle without the existence of their parent objects. For example, a school department *uses* teachers. Any teacher may belong to more than one department. And so, if a department ceases to exist, the teacher will still exist.

Aggregation


```
Function Department(Teacher, Student)
{
    this.teacher= Teacher;
    this.student= Student;
}

}
```

Composition

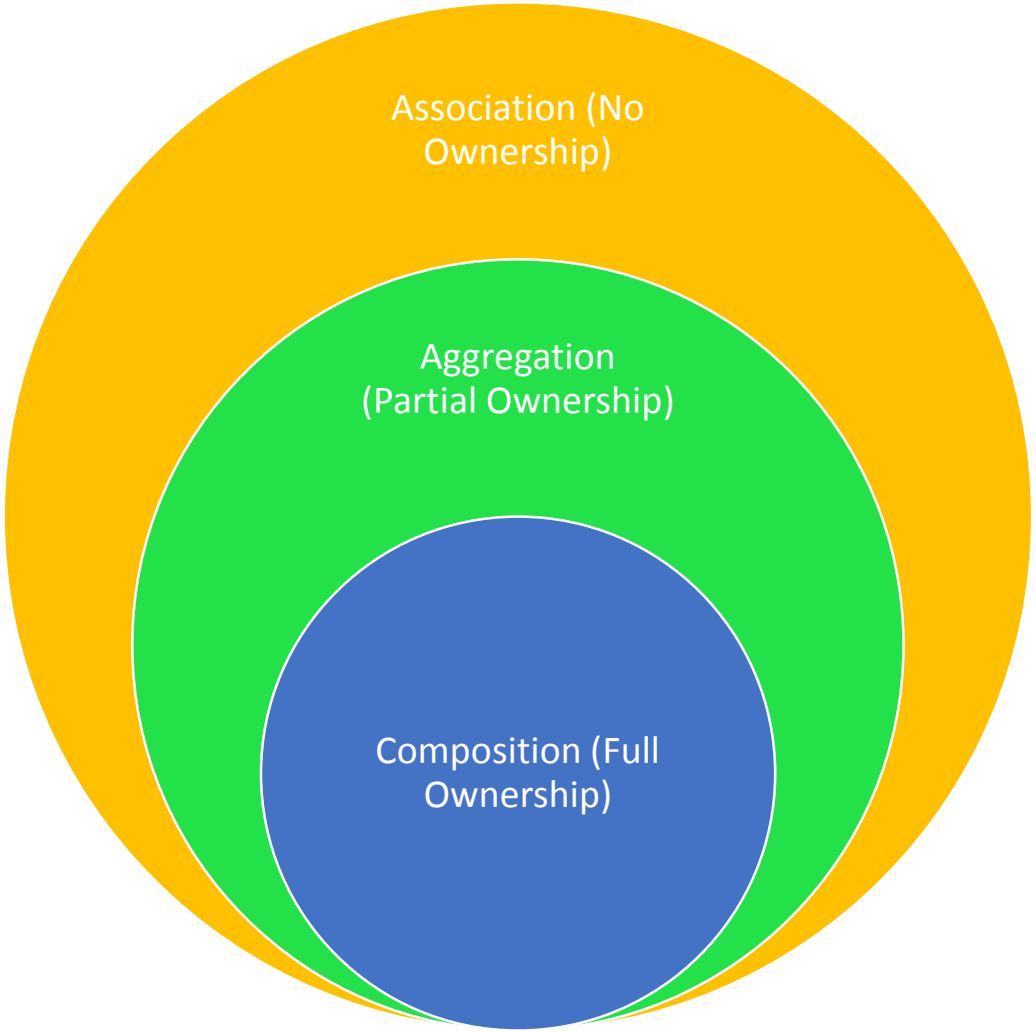
On the other hand, **Composition** is a *strong* type of **Association** with *full* ownership. This is strong compared to the weak Aggregation. For a **Composition** relationship, we use the term *owns* to imply a *strong* *has-a* relationship. For example, a department *owns* courses, which means that the any course's life-cycle depends on the department's life-cycle. Hence, if a department ceases to exist, the underlying courses will cease to exist as well.

Composition


```
Function Department(Teacher)
{
    this.teacher=Teacher;   // agregation
    this.course= new Course('English');   // composition
}
```

Difference between the 3

Digital Dreams



Abstraction

Abstraction is an emphasis on the idea, qualities and properties rather than the particulars (a suppression of detail). The importance of abstraction is derived from its ability to hide irrelevant details and from the use of names to reference objects. Abstraction is essential in the construction of programs. It places the emphasis on what an object is or does rather than how it is represented or how it works. Thus, it is the primary means of managing complexity in large programs.

Generalization

While abstraction reduces complexity by hiding irrelevant detail, generalization reduces complexity by replacing multiple entities which perform similar functions with a single construct. Generalization is the broadening of application to encompass a larger domain of objects of the same or different type.

Generalization

Programming languages provide generalization through variables, parameterization, generics and *polymorphism*. It places the emphasis on the similarities between objects. Thus, it helps to manage complexity by collecting individuals into groups and providing a representative which can be used to specify any individual of the group

Inheritance

The ability of a new class to be created, from an existing class by extending it, is called *inheritance*.

Javascript uses the .prototype to extend an object

Inheritance


```
function Foo ()
{
    tar:”okeke”;
}
 Foo.prototype.bar = true;
```

Polymorphism

- Polymorphisms is a generic term that means 'many shapes'. More precisely *Polymorphisms* means the ability to request that the same operations be performed by a wide range of different types of things.

Polymorphism

- At times, I used to think that understanding Object Oriented Programming concepts have made it difficult since they have grouped under four main concepts, while each concept is closely related with one another. Hence one has to be extremely careful to correctly understand each concept separately, while understanding the way each related with other concepts.

Polymorphism

- In *OOP* the *polymorphisms* is achieved by using many different techniques named
  - method overloading
  - method overriding

Method Overloading

Method overloading is the ability to define several methods all with the same name but with different arguments

The actual function that will be call will now depend on the number or type of argument supplied to the method

Method Overloading Example


```
public class MyLogger {
    public void LogError(Exception e) {
        // Implementation goes here
    }
    public bool LogError(Exception e, string message) {
        // Implementation goes here
    }
}
```

Method Overriding

Method overriding is a language feature that allows a subclass to override a specific implementation of a method that is already provided by one of its super-classes.

Method Overriding

- A subclass can give its own definition of methods but need to have the same signature as the method in its super-class. This means that when overriding a method the subclass's method has to have the same name and parameter list as the super-class' overridden method.

JSON

JSON is short for **JavaScript Object Notation**, and is *a way to store information in an organized, easy-to-access manner.* In a nutshell, it gives us a human-readable collection of data that we can access in a really logical manner.

- JSON data is written as name/value pairs.
- A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:
- **Example**

```
"firstName":"John"
```

JSON Values

- JSON values can be:
- A number (integer or floating point)
- A string (in double quotes)
- A Boolean (true or false)
- An array (in square brackets)
- An object (in curly braces)
- null

JSON Object

- JSON objects are written inside curly braces.
- Just like JavaScript, JSON objects can contain multiple name/values pairs:
- **Example**

```
{"firstName":"John", "lastName":"Doe"}
```

```
var Student= {"firstName":"John", "lastName":"Doe"};
```

JSON Array

- JSON arrays are written inside square brackets.
- Just like JavaScript, a JSON array can contain multiple objects:
- **Example**

```
"employees":[
   {"firstName":"John", "lastName":"Doe"},
   {"firstName":"Anna", "lastName":"Smith"},
   {"firstName":"Peter","lastName":"Jones"}
]
```

Parsing JSON

A common use of JSON is to read data from a web server, and display the data in a web page.

- Create a JavaScript string containing JSON syntax:

```
var text = '{ "employees" : [' +
'{ "firstName":"John" , "lastName":"Doe" },' +
'{ "firstName":"Anna" , "lastName":"Smith" },' +
'{ "firstName":"Peter" , "lastName":"Jones" } ]}';
```

Parsing JSON

The JavaScript function JSON.parse(*text*) can be used to convert a JSON text into a JavaScript object:

```
var obj = JSON.parse(text);
alert(obj.employees[1].firstName);
```

Looping through Objects

Objects can be loop through using the ‘for - in’ loop

```
 for (i in obj.employees)
{
    alert(obj.employees[i].firstName);
}
```

Looping through Objects


```
 for (i in obj.employees)
{
    var k=obj.employees[i];
    for(j in k)
    {
        alert(k[j]);
    }
}
```

Digital Dreams

?