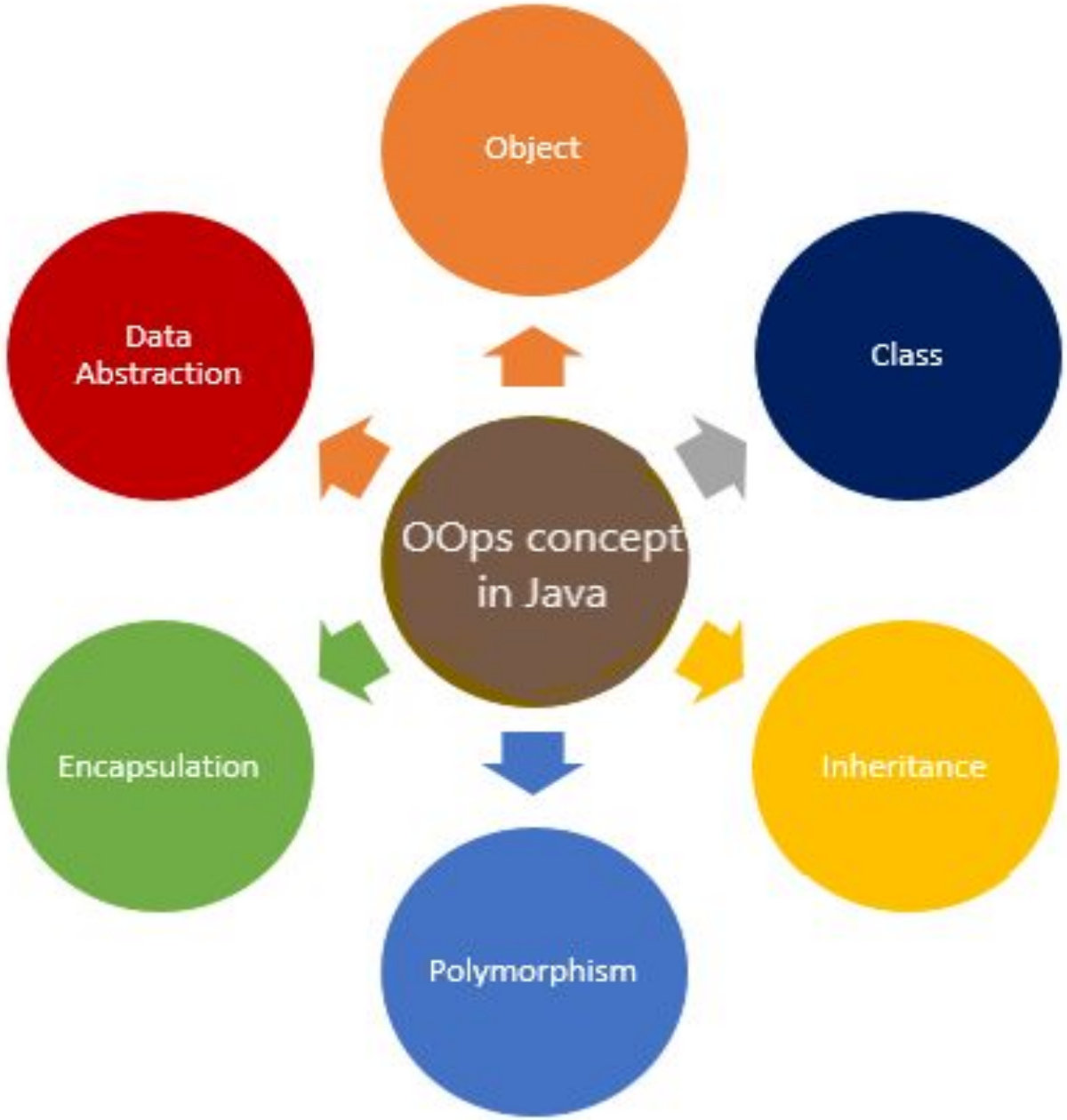


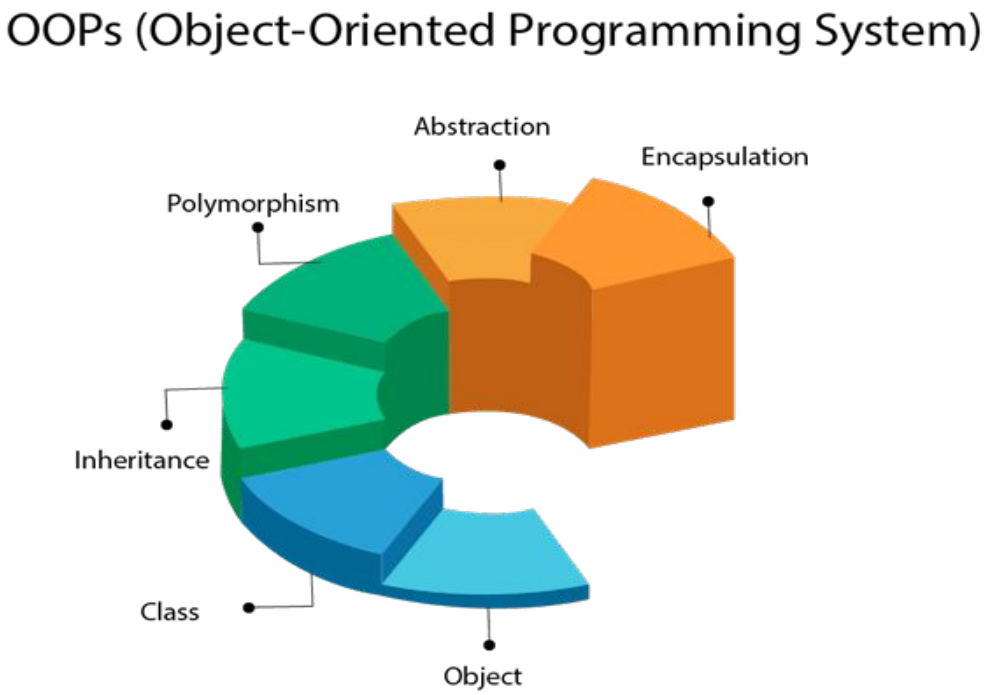

# CS20004: Object Oriented Programming using Java

**Lec-1**

**January 10, 2024**

# In this Discussion . . .

- Programming Paradigm
  - What a programming paradigm **is not**
  - Popular Programming Paradigms
    - Imperative
    - Procedure oriented
    - Functional
    - Declarative
    - Object oriented
  - Procedural Programming Vs. Object-oriented Programming
- OOP concept:
  - Class
- References



# What is a Programming Paradigm?

- Programming paradigms are different ways or styles in which a given program or programming language can be organized.
- Each paradigm consists of certain structures, features, and opinions about how common programming problems should be tackled.

# What is a Programming Paradigm? (Contd.)

- Programming paradigms are different ways or styles in which a given program or programming language can be organized.
- Each paradigm consists of certain structures, features, and opinions about how common programming problems should be tackled.

- The question of why are there many different programming paradigms is similar to why are there many programming languages.
- Certain paradigms are better suited for certain types of problems, so it makes sense to use different paradigms for different kinds of projects.

# What is a Programming Paradigm? (Contd.)

- The practices that make up each paradigm have developed through time.
  - Thanks to the advances both in software and hardware, different approaches have come up that didn't exist before.
- And at last, there's human creativity.
  - As a species, we just like creating things, improving what others have built in the past, and adapting tools to our preference or to what seems more efficient to us.
- All this results in the fact that today we have many options to choose from when we want to write and structure a given program.

# What a Programming Paradigm **is not**

- Programming paradigms are not languages or tools.
  - We can't "build" anything with a paradigm.
  - They’re more like a set of ideals and guidelines that many people have agreed on, followed, and expanded upon.

# What a Programming Paradigm **is not (Contd.)**

- Programming languages aren't always tied to a specific paradigm.
  - There are languages that have been built with a certain paradigm in mind and have features that facilitate that kind of programming more than others (Haskel and functional programming is a good example).

# What a Programming Paradigm **is not (Contd.)**

- But there are also "multi-paradigm" languages, meaning we can adapt our code to fit a certain paradigm or another (JavaScript and Python are good examples).
- At the same time, programming paradigms aren't mutually exclusive, in the sense that we could use practices from different paradigms at the same time with no problem at all.

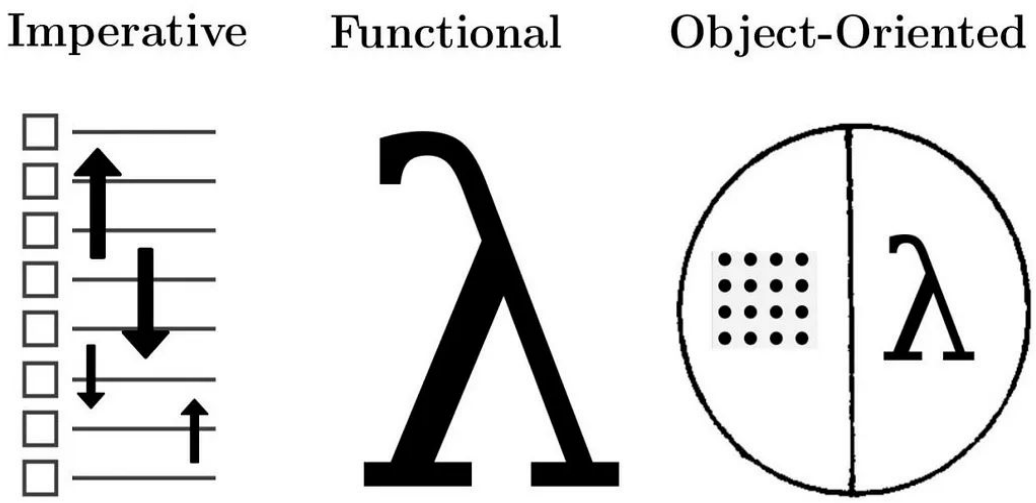
# Why should I Care?

# Why should I Care?

- **Short answer:** general knowledge.
- **Long answer:** It's interesting to understand the many ways in which programming can be done. Exploring these topics is a good way of opening our mind and helping us think outside the box and outside the tools we already know.
- Moreover, these terms are used a lot in the coding world, so having a basic understanding will help us better understand other topics as well.

# Popular Programming Paradigms

- Keep in mind this list is not exhaustive. There are other programming paradigms not covered here, although we will be covering the most popular and most widely-used ones.
  - Imperative Programming
  - Procedural Programming
  - Functional Programming
  - Declarative Programming
  - Object-Oriented Programming



# Imperative Programming

- Imperative programming consists of sets of detailed instructions that are given to the computer to execute in a given order.
  - It's called "**imperative**" because as programmers we dictate exactly what the computer has to do, in a very specific way.
  - Imperative programming focuses on describing how a program operates, step by step.

# Imperative Programming (Contd.)

- For instance, suppose we want to bake a cake. Then, imperative program to do this might look like the one's written in the box:

I'm not a great cook, so


1- Pour flour in a bowl
2- Pour a couple eggs in the same bowl
3- Pour some milk in the same bowl
4- Mix the ingredients
5- Pour the mix in a mold
6- Cook for 35 minutes
7- Let chill

# Imperative Programming (Contd.)

- Using an actual code example, let's say we want to filter an array of numbers to only keep the elements bigger than 5. Our imperative code might look like this:

```
const nums = [1,4,3,6,7,8,9,2]
const result = []

for (let i = 0; i < nums.length; i++)
{
                if    (nums[i]    >    5)
result.push(nums[i])
}


 console.log(result) // Output: [ 6,
 7, 8, 9 ]
```

- See that we're telling the program to iterate through each element in the array, compare the item value with 5, and if the item is bigger than 5, push it into an array.

- We're being detailed and specific in our instructions, and that's what imperative programming stands for.

# Procedural Programming

- Procedural programming is a derivation of imperative programming, adding to it the feature of functions (also known as "procedures" or "subroutines").
- In procedural programming, the user is encouraged to subdivide the program execution into functions, as a way of improving modularity and organization.

# Procedural Programming (Contd.)

- Following our cake example, procedural programming may look like this:

```
function pourIngredients() {
    - Pour flour in a bowl
    - Pour a couple eggs in the same bowl
    - Pour some milk in the same bowl
}

function mixAndTransferToMold() {
    - Mix the ingredients
    - Pour the mix in a mold

  }
```

```
function cookAndLetChill() {
    - Cook for 35 minutes
    - Let chill
}

pourIngredients()
mixAndTransferToMold()
cookAndLetChill()
```

# Procedural Programming (Contd.)

- It can be seen that, thanks to the implementation of functions, we could just read the three function calls at the end of the file and get a good idea of what our program does.
- That simplification and abstraction is one of the benefits of procedural programming. But within the functions, we still got same old imperative code.

# Functional Programming

- Functional programming takes the concept of functions a little bit further.
  - In functional programming, functions are treated as **first-class citizens**, meaning that they can be assigned to variables, passed as arguments, and returned from other functions.

# Functional Programming (Contd.)

- Another key concept is the idea of **pure functions**.
  - A pure function is one that relies only on its inputs to generate its result.
  - And given the same input, it will always produce the same result.
  - Besides, it produces no side effects (any change outside the function's environment).

# Functional Programming (Contd.)

- With these concepts in mind, functional programming encourages programs written mostly with functions.
- It also defends the idea that code modularity and the absence of side effects makes it easier to identify and separate responsibilities within the codebase.
- This therefore improves the code maintainability.

# Functional Programming (Contd.)

- Going back to the array filtering example, we can see that with the imperative paradigm we might use an external variable to store the function's result, which can be considered a side effect.

```
const nums = [1,4,3,6,7,8,9,2]
const result = [] // External variable

for (let i = 0; i < nums.length; i++) {
    if (nums[i] > 5) result.push(nums[i])
}


  console.log(result) // Output: [ 6, 7,

  8, 9 ]
```

Corresponding Functional Programming

```
const nums = [1,4,3,6,7,8,9,2]

function filterNums() {
    const result = [] // Internal variable

    for (let i = 0; i < nums.length; i++) {
        if (nums[i] > 5) result.push(nums[i])
    }

    return result
}


  console.log(filterNums()) // Output: [ 6, 7,

  8, 9 ]
```

# Functional Programming (Contd.)

- It's almost the same code, but we wrap our iteration within a function, in which we also store the result array.
  - In this way, we can assure the function doesn't modify anything outside its scope.
  - It only creates a variable to process its own information, and once the execution is finished, the variable is gone too.

# Declarative Programming

- Declarative programming is all about hiding away complexity and bringing programming languages closer to human language and thinking.
  - It's the direct opposite of imperative programming in the sense that the programmer doesn't give instructions about how the computer should execute the task, but rather on what result is needed.

# Declarative Programming (Contd.)

- As an example, the same array filtering story, a declarative approach might be:

| | |
|---|---|
| `const nums = [1,4,3,6,7,8,9,2]`<br><br>`console.log(nums.filter(num => num > 5))`<br>`// Output: [ 6, 7, 8, 9 ]` | See that with the filter function, we're not explicitly telling the computer to iterate over the array or store the values in a separate array. We just say what we want ("filter") and the condition to be met ("num > 5"). |

What's nice about this is that it's easier to read and comprehend, and often shorter to write. JavaScript's filter, map, reduce and sort functions are good examples of declarative code.

# Declarative Programming (Contd.)

- An important thing to notice about declarative programming is that under the hood, the computer processes this information as imperative code anyway.
- Following the array example, the computer still iterates over the array like in a for loop, but as programmers we don't need to code that directly.
  - What declarative programming does is to hide away that complexity from the direct view of the programmer.

# Object-Oriented Programming

- One of the most popular programming paradigms is object-oriented programming (OOP).
- The **core** concept of OOP is to separate concerns into entities which are coded as objects. Each entity will group a given set of information (properties) and actions (methods) that can be performed by the entity.

# Object-Oriented Programming (Contd.)

- **OOP makes heavy usage** of classes (which are a way of creating new objects starting out from a blueprint or boilerplate that the programmer sets).
- Objects that are created from a class are called **instances.**

# Examples of Classes and Objects

| | |
|---|---|
| 1. **Class** - Human being<br>Man - an object of Human being<br>Woman - an object of Human being | 3. **Class** - pets<br>Dog - an object of pets<br>Cat - an object of pets |
| 2. **Class** - colour<br>Red - an object of colour<br>Blue - an object of colour | 4. **Class** - Wild animals<br>Tiger - an object of Wild animals<br>Lion - an object of Wild animals |

# Examples of Classes and Objects (Contd.)

| | |
|---|---|
| 5. **Class** - drinks<br>Hot drink - an object of drinks<br>Cold drink - an object of drinks | 7. **Class**- Expensive Cars<br>Objects- Mercedes, BMW, Toyota |
| 6. **Class** - pen<br>Ballpoint pen - an object of pen<br>Ink pen - an object of pen. | 8. **Class** - Subject<br>OOPS with Java - an object of Subject<br>Operating Systems - an object of Subject |

# Object-Oriented Programming (Contd.)

- Following our pseudo-code cooking example, now let's say in our bakery we have a main cook (called Frank) and an assistant cook (called Anthony) and each of them will have certain responsibilities in the baking process.
- If we used OOP, our program might look like this (See next slide).

# Object-Oriented Programming (Contd.)

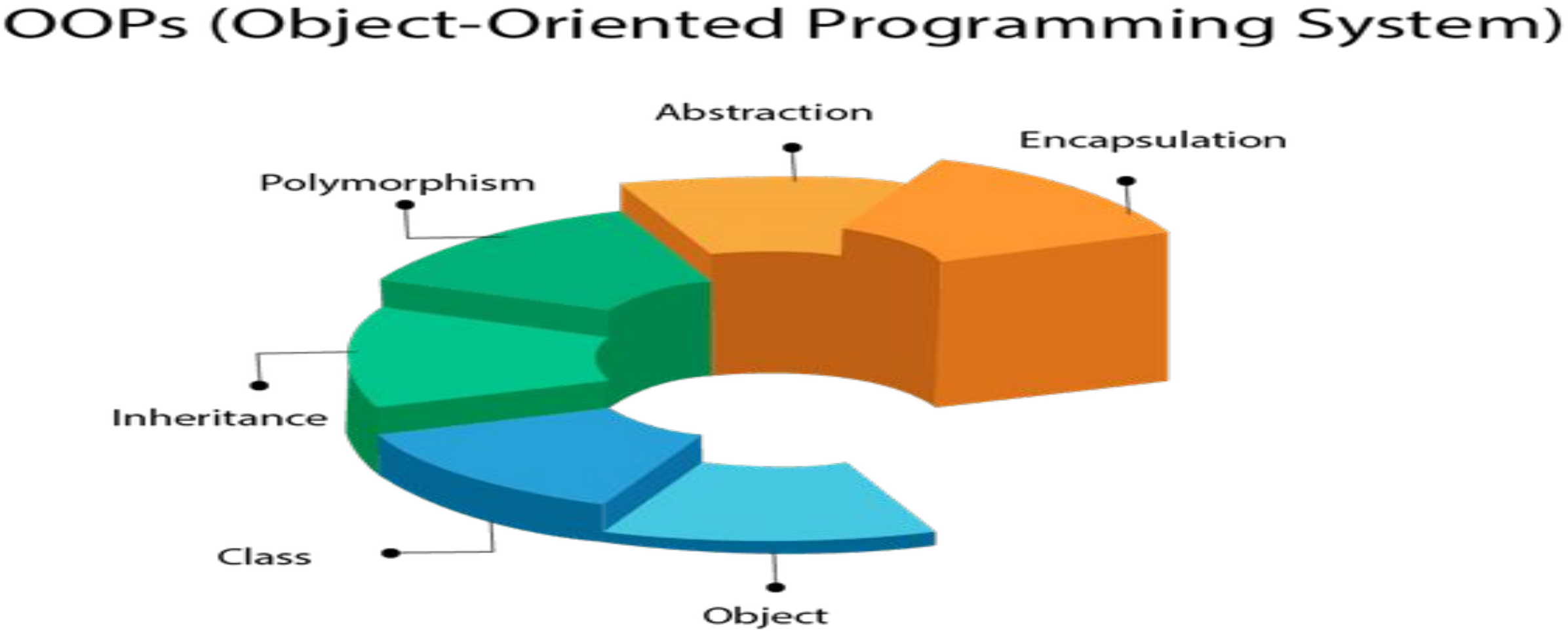
```
// Create the two classes corresponding to each entity
class Cook {
        constructor constructor (name) {
        this.name = name
    }

    mixAndBake() {
        - Mix the ingredients
        - Pour the mix in a mold
        - Cook for 35 minutes
    }

    }


    // Instantiate an object from each class


    const Frank = new Cook('Frank')


    const Anthony = new AssistantCook('Anthony')
```

```
class AssistantCook {
    constructor (name) {
        this.name = name
    }

    pourIngredients() {
        - Pour flour in a bowl
        - Pour a couple eggs in the same bowl
        - Pour some milk in the same bowl
    }
    chillTheCake() {
        - Let chill
    }
}

    // Call the corresponding methods from each instance


    Anthony.pourIngredients()


    Frank.mixAndBake()


    Anthony.chillTheCake()
```

# Object-Oriented Programming (Contd.)

- What's nice about OOP is that it facilitates the understanding of a program, by the clear separation of concerns and responsibilities.
- In the example in the previous slide we've just scratched the surface of the many features of OOP.

# Object-Oriented Programming (Contd.)



# Procedural Programming Vs. Object Oriented Programming

- The **objective** of procedural programming is to break down a program into a collection of variables, data structures whereas the **main aim** of object-oriented programming is to break down a programming task into objects.
  - In simple words, **procedural programming uses procedures to operate on data structures**, while **object-oriented uses objects for the purpose**.

# Procedural Programming Vs. Object Oriented Programming

| Parameter | Procedural Programming | Object-Oriented Programming |
|---|---|---|
| Definition | In procedural programming, we break down a task into variables and routines. After that, it is carried out every step one by one. | In Object-Oriented Programming, a programming model is based upon the concepts of objects, or where everything is represented as an object. |
| Security | Procedural programming is less secure than Object-Oriented Programming. | Object-Oriented Programming is secure because data hiding is possible in OOPs due to abstraction. |

# Procedural Programming Vs. Object Oriented Programming (Contd.)

| Parameter | Procedural Programming | Object-Oriented Programming |
|---|---|---|
| Approach | Procedural programming follow top-down approach | Object-Oriented Programming follow bottom-up approach |
| Access Modifiers | No access modifiers | Private, Public, and Protected are the access modifiers |
| Code Reusability | No code reusability | Due to Inheritance, the concept of code reusability exists |

# Procedural Programming Vs. Object Oriented Programming (Contd.)

| Parameter | Procedural Programming | Object-Oriented Programming |
|---|---|---|
| Program Division | the program is divided into small functions. | a program is divided into small objects. |
| Problem Size | not ideal for solving large or complex problems. | suitable for solving large or complex problems. |
| Examples | Examples of procedural programming are FORTRAN, ALGOL, COBOL, BASIC, Pascal, and C. | Examples of object-oriented programming are Java, C++, C#, Python, PHP, JavaScript, Ruby, Perl, Objective-C, Dart, Swift, and Scala. |

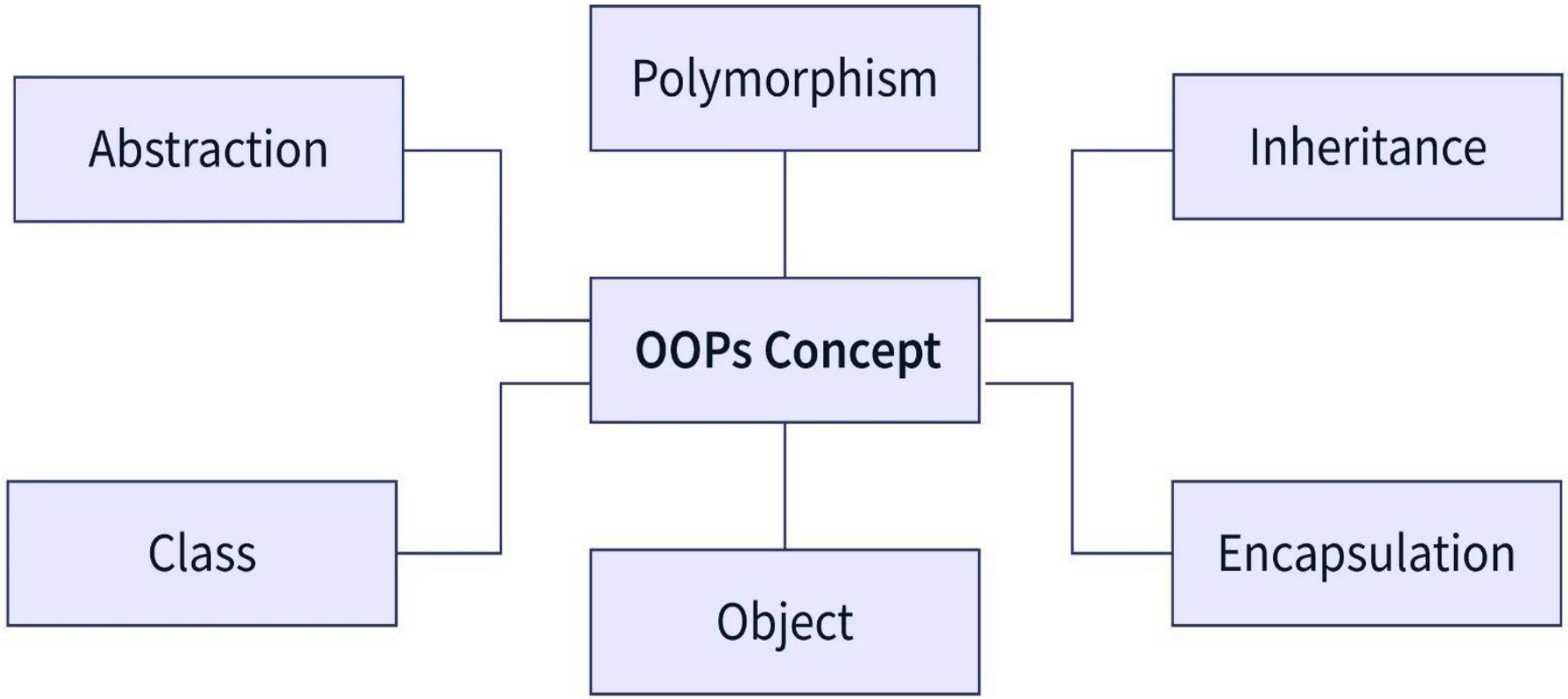
# Procedural Programming Vs. Object Oriented Programming (Contd.)

| Parameter | Procedural Programming | Object-Oriented Programming |
|---|---|---|
| Access to Data | most functions use global data for sharing and can freely access it from one function to another. | data cannot move easily from one function to another. It can be kept private or public, allowing user control over data access. |
| Data Sharing | It shares global data among the functions in the program. | It shares data among objects through its member functions. |

# OOPS Concept

- As the name suggests, objects in programming.
  - A programming model is based upon the concepts of objects, or where everything is represented as an object.
  - In OOP, we try to see the whole world as an object or in terms of objects.
  - **For example**, dogs have states or data members ***like*** color, hungry, and breed, **and** the behaviors or methods (actions they can perform) are barking, wagging their tail, eating, sleeping, etc.

# OOPS Concept



# References


1. https://www.freecodecamp.org/news/an-introduction-to-programming-paradigms/
2. https://www.haskell.org/
3. https://betterprogramming.pub/programming-paradigms-object-oriented-vs-procedural-f89eda302b0e
4. https://www.codingninjas.com/studio/library/difference-procedural--object-oriented-programming
5. https://www.scaler.com/topics/cpp/procedural-programming/
6. https://testbook.com/key-differences/difference-between-procedural-and-object-oriented-programming
7. https://zriyansh.medium.com/real-life-oop-examples-b77ce3ac228b
8. https://www.mygreatlearning.com/blog/oops-concepts-in-java/
9.