CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY (A), HYDERABAD -75

Department of Computer Science and Engineering

BE (CSE) II-Semester Object Oriented Programming (22CSC03)

Unit-1: **Introduction to OOPs**- Programming Paradigms, advantages of OOP, comparison of OOP with Procedural Paradigms; **Classes and Objects**: Prototyping, referencing the variables in functions, inline, static functions, memory allocation for classes and objects, arrays of objects, constructors.

Programming Paradigms

Programming paradigm: is the way/style/approach/methodology in which a given program or programming language can be organized. There are different programming paradigms. Each paradigm consists of certain structures, features, and opinions about how common programming problems should be tackled.

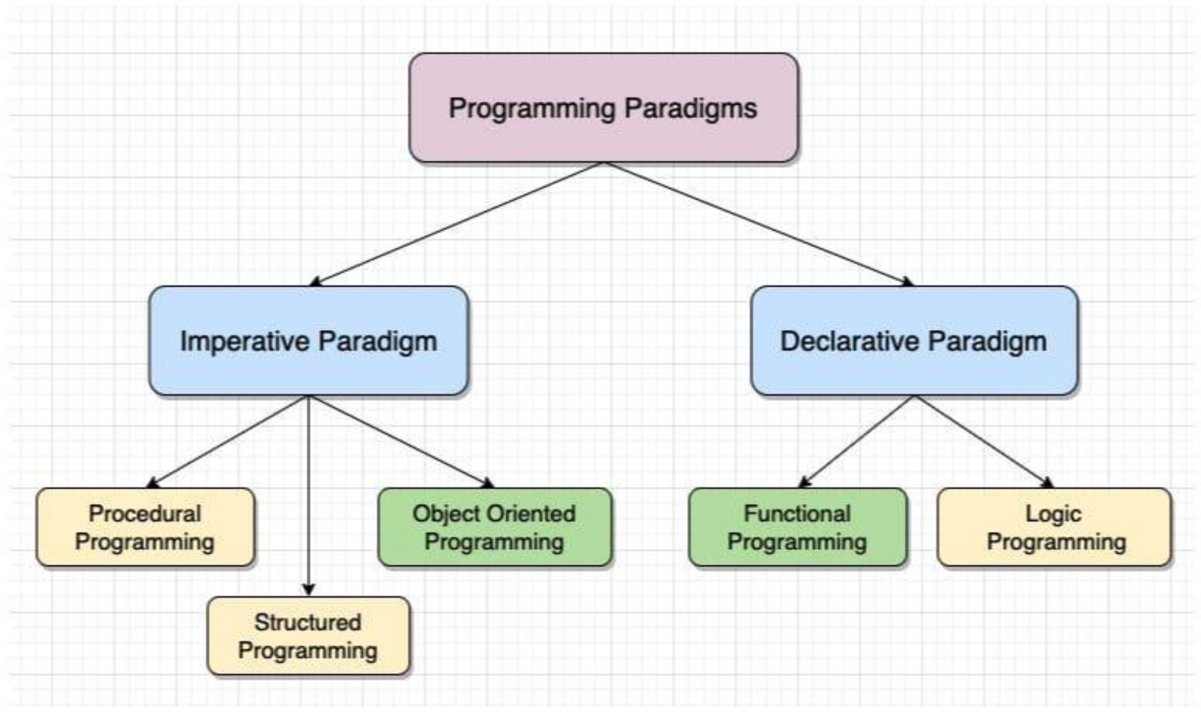

Figure 1: Programming Paradigms

Popular Programming Paradigms

Two main approaches

- ***Imperative programming***: focuses on how to execute, defines *control flow as statements* that change a program state. Direct assignment, common data structures, global variables.
- ***Declarative programming***: focuses on what to execute, defines *program logic*, but not detailed control flow.

There are several programming paradigms. Most widely used programming paradigms are:

- **Imperative Programming**
  - Structured Programming
  - Procedural Programming
  - Object-Oriented Programming
- **Declarative Programming**
  - Functional Programming
  - Logic Programming

Imperative Programming: Consists of sets of detailed instructions that are given to the computer to execute in a given order. It is 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. Suppose, we want to bake a cake, the imperative program to do this might look like:

```
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
```

Actual code example: let us say, we want to filter an array of numbers to only keep the elements bigger than 6. The 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] >6) result.push(nums[i])
}

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

Here, we are instructing the computer to

- Iterate through each element in the array,
- Compare the item with 6, and if the item is bigger than or equal to 6, push it into an array.

Structured Programming (1950's): A style of imperative programming with more logical program structure. Aimed to improve clarity, quality, and development time of a computer program by making extensive use of the structured control flow constructs of selection (if/then/else) and repetition (while and for), block structures, and subroutines. Follows indentation. No or limited use of goto statements. Ex: *C*, *C++*, *Java*, *Kotlin*, *Pascal*, *PHP*, *Python*.

Procedural Programming (1957-72): Classified as imperative programming, that involves the implementation of computer programs as procedures (functions/ Procedures/ Subroutines) that call each other. Thus the program is a series of steps that forms a hierarchy of calls to its constituent procedures. Procedural programming encourages the programmers to divide the programs into smaller units for improving modularity and efficient organization. Example: FORTRAN, ALGOL, COBOL, BASIC, Pascal, *C*.

```
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()
```

In the above implementation, three procedures/functions are defined and called later in the last three lines.

Benefits

- Complexity of the problem can be simplified.
- Provides abstraction. (implementation details can be hidden)
- Easy to debug and modify in case of errors
- Same module (function/procedure) can be used several times at different places.

Object-Oriented Programming (OOP) –mid 1990's: OOP is one of the most popular programming paradigms. The main features of object-oriented programming are:

- Data encapsulation
- Abstraction
- Inheritance
- Polymorphism

Object Oriented Programming paradigm is based on the concept of entities called ***objects***. Each entity contain data and code. The data is in the form of fields /properties (also called ***attributes***) and the code is in the form of procedures/methods (also called ***behavior***). Programs are designed by making use of objects that interact with one another.

OOP makes heavy usage of ***classes*** (blue prints) and ***objects*** (also called instances) that are created from a class. Ex: *Common Lisp, C++, C#, Eiffel, Java, Kotlin, PHP, Python, Ruby, Scala, JavaScript.*

```
// 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
    }
}

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
    }
}
// Instantiate an object from each class
const Frank = new Cook('Frank')
const Anthony = new AssistantCook('Anthony')

// Call the corresponding methods from each instance
Anthony.pourIngredients()
Frank.mixAndBake()
Anthony.chillTheCake()
```

Declarative Programming

Declarative programming paradigm is a style of building the structure and elements of computer programs. The program elements express the logic of a computation without describing its detailed control flow.

Declarative programming hides the complexity and brings the programming languages closer to human language and thinking. Declarative programming is opposite to imperative programming where the programmer need not give the instructions about how the computer execute a particular task, but here, needs to tell what result is required.

For example, to filter an array whose elements are greater than 6, the declarative code will look like:

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

console.log(nums.filter(num => num > 6)) // Output: [ 7, 8, 9 ]
```

Good examples: JS frameworks/libraries like React.

```
<button onClick={() => console.log('You clicked me!')}>Click me</button>
```

In the above implementation, we have a button element, with a listener that fires a console.log function when the button is clicked.

- React uses JSX syntax, which mixes HTML and JS in the same thing.
- It is easier and faster to write Apps but the browser does not read and execute. The React code is later transformed into regular HTML and JS to run by the browser.
- JSX is declarative, in the sense that its purpose is to give developers a friendlier and more efficient interface to work with.
- Computer processes the information as imperative way.
- Declarative programming hides the complexity from programmer.

Functional Programming: Programs are constructed by applying and composing functions. It is a declarative programming paradigm where the functions like mathematical expressions that map values to other values. It takes the concept of functions a little bit further. Here functions are treated as first-class citizens as they can be bound to names, assigned to variables, parameters can be passed as arguments, and values can be returned.

Another key concept of functions is **pure functions**. If a function relies only on its inputs and produces its result, then it is called a "pure function". Further, pure functions always generate the same result for the same inputs and produces n side effects. They don't change outside environment. Ex: *Lisp*, *Kotlin*, *Go*, *Rust*,*Python*, *R*, *Ruby*, *Scala*, *JavaScript*.

Functional programming

- Programmers always write programs mostly with functions.
- Defends the idea of code modularity.
- There are no commands, only side-effect free expressions
- Code is much shorter, less error-prone and much easier to debug.
- There is more inherent parallelism, good compilers can produce faster code.
- It is easier to identify and separate the responsibility within the codebase
- Improves the code maintainability

For example, to filter the elements of an array, where the element are greater than 6:

```
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] > 6) result.push(nums[i])
    }

    return result
}

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

In the above implementation, we can ensure that the function can't modify anything outside its code. It only creates variables locally and process the information. At the end, once the execution is completed, then the local variables are destroyed.

Logic Programming paradigm: Makes us of sentences that follow logic that they express facts and rules. Computation is performed by making logical inferences on all available data. The computer program that use this paradigm must have a base of existing logic, called '*predicates*'. The predicates are used to form atoms or atomic formulas, which states true. Logic languages often rely on queries to display the relevant data. Queries are part of Machine learning, where they run automatically without human intervention.

There are different logic programming languages. The most common logic programming is Prolog created in 1970s. Prolog utilizes AI that helps to form conclusions and quickly process large amounts of data. Prolog helps businesses and organizations through:

- ***Natural Language Processing (NLP)***: Allows better interactions between humans and computers. Listen human language in real-time, then process and translate it for computers. NLP used with both spoken languages as well as documents that are printed or application-oriented. Amazon Alexa, Google Home use NLP use NLP to process, understand spoken instructions, as well as email applications to filter spam emails and warn phishing attempts.
- ***Database management***: Logic programming makes it easy to create, maintain, querying NoSQL databases. Using logic programming, we can create databases out of big data, identify the relevant information, query and store it in the appropriate area.
- ***Predictive Analytics***: Logic languages helps to search for inconsistencies or areas of differentiation for making predictions with large datasets.

Logic programming can be

- Higher order,
- Constraint logic,
- Concurrent logic,
- Abductive logic,
- Inductive logic etc.

```
predicates
  sumoftwonumber(integer, integer).

clauses
  sumoftwonumber(0, 0).
  sumoftwonumber(N, R) :-
    N > 0,
    N1 is N - 1,
    sumoftwonumber(N1, R1),
    R is R1 + N.
```

Advantages of OOP: Object Orient Programming offers several advantages. Fe important advantages are:

- **Security**: Many developers use OOP because it ensures minimal exposure using encapsulation where they bundle data to encapsulate information inside an object. . it makes the code secure and free of unintended data corruption.
- **Improved collaboration**: Allows developers to divide complex software systems into small, manageable objects. Each object is responsible for a specific function. They can develop, test, and maintain these self-contained units independently. Thus, it helps to organize code and streamlining collaboration when necessary.
- **Reuse of code**: The *inheritance* property of OOP allows code reuse. Developers can create classes based on existing ones. Thus, it reduces code duplication and saves development time significantly. Developer need not have to write the same code multiple times. In essence, it creates enough space for through data analysis.
- **Makes changes seamlessly**: Abstraction is one of the pivotal benefit of OOP, through which complex systems are modeled into simple classes. The modeling of classes is done on essential *properties* and *behaviors* relevant to the problem. In fact, objects only reveals mechanisms, and the focus is on essential features. It is easy for the developers to make additional changes to the code or incorporate additions over time.
- **Easy location and fixing of bugs**: It is one of the key advantage of OOP. Developers can isolate and test each object for identifying and isolating the issues. It is also easy to trace the source of issue to make the changes or improvements.
- **Flexibility**: The polymorphism concept of OOP, allows a developer to treat different types of objects as objects of a common class. It helps to eliminate the code duplication by using the existing code in a flexible way without the need of knowing their specific class.
- **Productivity**: Top most benefit of OOP is the productivity, where developers can mount big projects without any hiccups. It helps developers to improve software quality at an affordable cost.
- **Code consistency**: As it models real-world objects, the code is more readability and maintainable. The OOP features makes it convenient to maintain and update the code. Developers can add or change the features easily to meet the requirements. The change of code does not affect the entire codebase and it also facilitate debugging.
- **Scalability**: The features of OOP allows the developers to scale the software systems into complex systems in a smooth way. Developers can add the new code in a hassle-free manner.
- **Reduced development cost**: This is another key benefit of OOP. Developers can reuse libraries of classes multiple times as per the requirement. This reduces the development.

Drawbacks:

- Over emphasis on data component of software.
- Lack of focus on computation or algorithms.
- Causes code-writing complications, and compiling takes time.

Comparison of Object-Oriented Programming with Procedural Paradigms

Procedural Programming	**Object-Oriented Programming**
Programs are divided into small parts called **functions / Procedures**	Programs are divided into small parts called **objects**
Follows **top-down approach**	Follows **bottom-up approach**
Adding new data and functions is not easy	Easy
Procedural programming does not have any proper way of hiding data so it is **less secure**	Provides data hiding so it is **more secure**
Overloading is not possible	Possible
No concept of data hiding and inheritance	Supports
Procedural programming based on the unreal world	Based on the real-world
Used for designing medium-sized programs	Large and complex programs
Uses the concept of procedure abstraction	Uses the concept of data abstraction
Code reusability absent in procedural programming	Code reusability is present
Ex: C, FORTRAN, Pascal, Basic, etc.	*C++, Java, Python, C#, Scala, Swift, etc.*

References

1. https://www.freecodecamp.org/news/an-introduction-to-programming-paradigms/
2. https://cs.lmu.edu/~ray/notes/paradigms/
3. https://www.geeksforgeeks.org/introduction-of-programming-paradigms/
4. https://www.geeksforgeeks.org/differences-between-procedural-and-object-oriented-programming/
5. https://www.linode.com/docs/guides/logic-programming-languages/