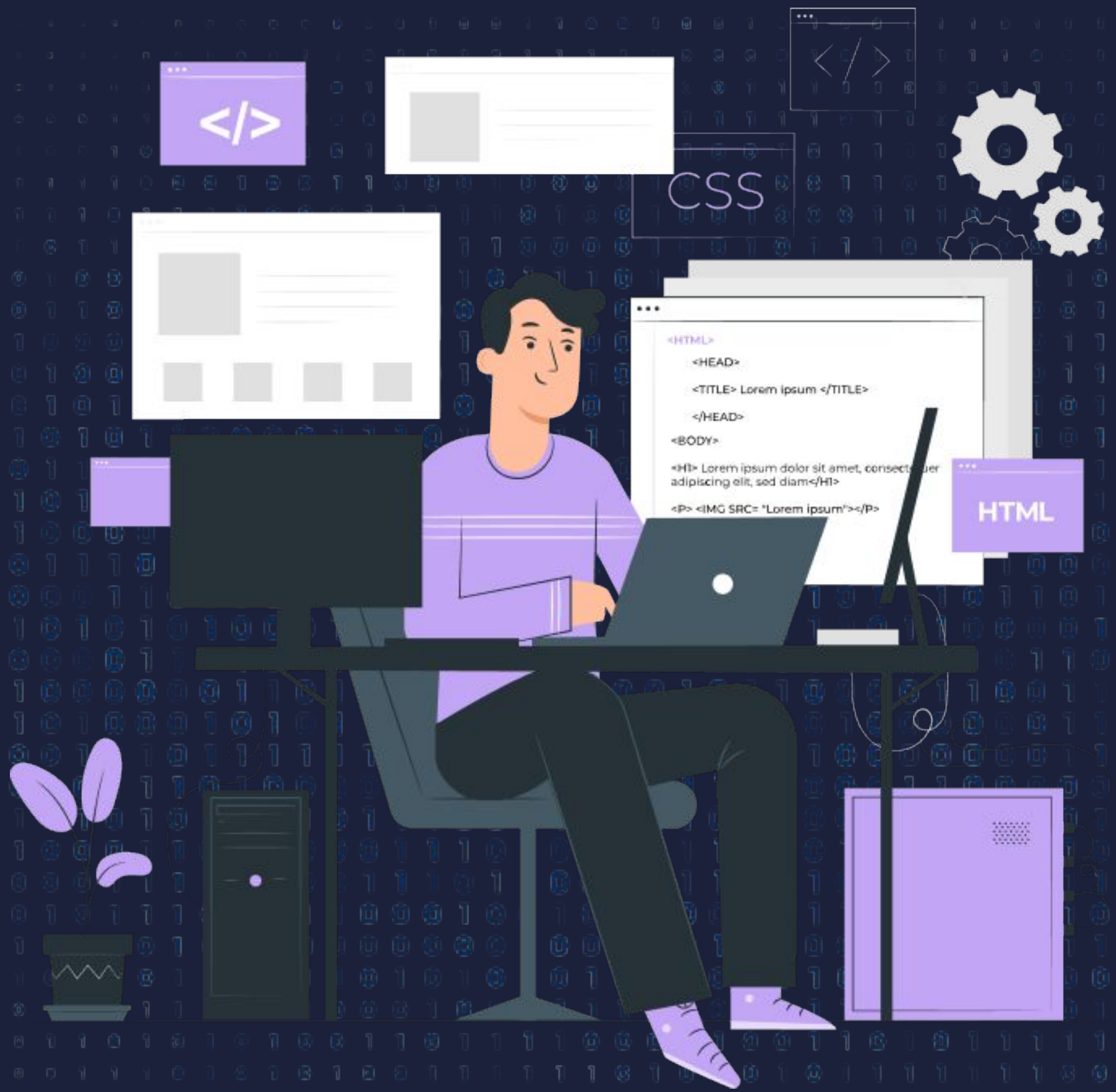
Pillars of Object-Oriented Programming - Encapsulation



Topics

- Introduction to Encapsulation
- Code Example of Encapsulation

Introduction to Encapsulation

- Encapsulation can be defined as the process which involves packing or bundling data (attributes and properties) and functions/methods (behaviour or operations) into one component and then controlling access to that component to make a black box out of the object. Because of this, the user of that class only needs to know its interface which refers to the set of public methods and properties that a class exposes to its users, and the private information remains hidden.

- Encapsulation is a fundamental concept in object-oriented programming (OOP), along with inheritance and polymorphism.

Some of the benefits of Encapsulation include -

- Information or data Protection and hiding- it provides a way to protect or hide the internal state of an object by hiding its data from direct access. By encapsulating data within an object and exposing it only through controlled methods (getters and setters).
- Modularity and Code organisation - it helps in creating modular and organized code by bundling related data and behaviour into objects, you can encapsulate the internal workings of an object, hiding its implementation details
- Code Reusability - it supports code reusability By encapsulating data and behaviour within objects, you can create reusable components. Objects can be instantiated and used in different parts of a program or in different programs altogether.
- Flexibility and Maintainability - it promotes flexibility and maintainability. By encapsulating data and behaviour within objects, you can change the internal implementation of an object without affecting other parts of the program. As long as the object's interface remains the same, other objects using it won't be impacted

Code Example of the Encapsulation

```
class Course {
  #name;
  #description;
  #category;
  #price;
  constructor(name, description, category, price) {
    // private properties --
    this.#name = name;
    this.#description = description;
    this.#category = category;
    this.#price = price;
  }
  // public methods
  getCourseName() {
    return this.#name;
  }
  getCourseDescription() {
    return this.#description;
  }
  getCourseCategory() {
    return this.#category;
  }
  getCoursePrice() {
    return this.#price;
  }}
```

```
const webCourse = new Course(
  "React",
  "This is react course desc",
  "FrontEnd",
  599
);
// console.log(webCourse.getCourseName());
console.log(webCourse.getCourseDescription());
console.log(webCourse.getCourseCategory());
console.log(webCourse.getCoursePrice());

console.log(webCourse.#name);//syntax error:private
field

/**
 ****** output *****
 React
This is react course desc
FrontEnd
599
 */
```

continue

From the above example -

- the **Course** class encapsulates the name, description, category and price properties as private members.
- The public methods **getCourseName(), getCourseDescription(), getCourseCategory(), and getCoursePrice()** provide access to these private members through the class's interface.
- It is a syntax error to refer to #name outside of the class.

SKILLS

THANK YOU