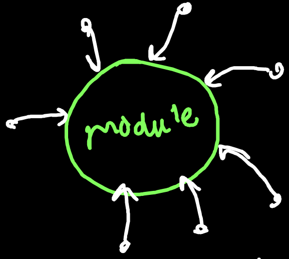
Q) What do you mean by $\boxed{\text{ABSTRACTION}}$ in Java? What are the advantages? How to achieve/implement abstraction?

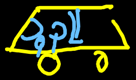
↳ Complexity ↓, redundancy ↓, consistency ↑

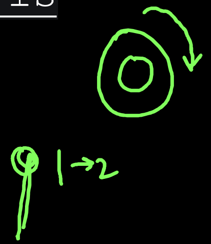


complexity ↓
essential info ↑↑

Data abstraction is the process of hiding certain details and showing only essential information to the user. Abstraction can be achieved with either abstract classes or interfaces.

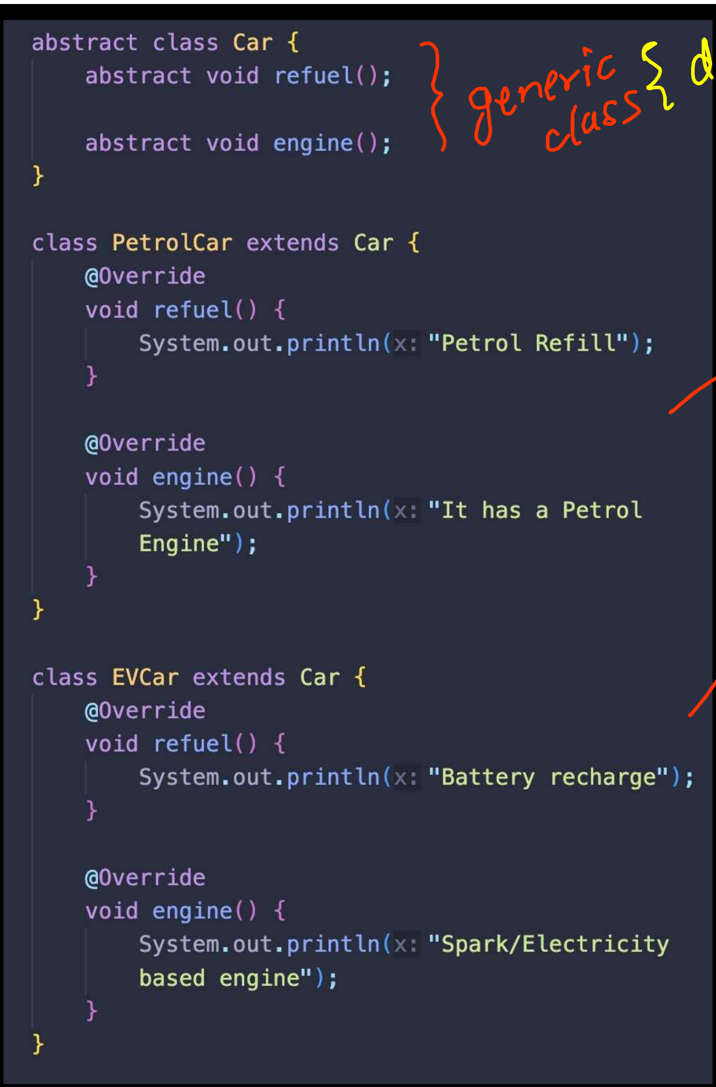
Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of a car or applying brakes will stop the car, but he does not know how on pressing the accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of the accelerator, brakes, etc in the car. This is what abstraction is.





Q) What is an abstract class? What are abstract methods? How is abstract class different from concrete class?

- abstract class cannot be instantiated
- abstract class may have some abstract methods.
  - method with only function prototype (access modifier, return type, function name, argument list) but no body
- abstract class is a more generic class whereas concrete class is a more specific class.
- Concrete class (Implementation) depends upon abstract class (abstraction)



```java
abstract class Car {
    abstract void refuel();

    abstract void engine();
}

class PetrolCar extends Car {
    @Override
    void refuel() {
        System.out.println(x: "Petrol Refill");
    }

    @Override
    void engine() {
        System.out.println(x: "It has a Petrol
        Engine");
    }
}

class EVCar extends Car {
    @Override
    void refuel() {
        System.out.println(x: "Battery recharge");
    }

    @Override
    void engine() {
        System.out.println(x: "Spark/Electricity
        based engine");
    }
}
```

} generic class { doesn't exist in real life }

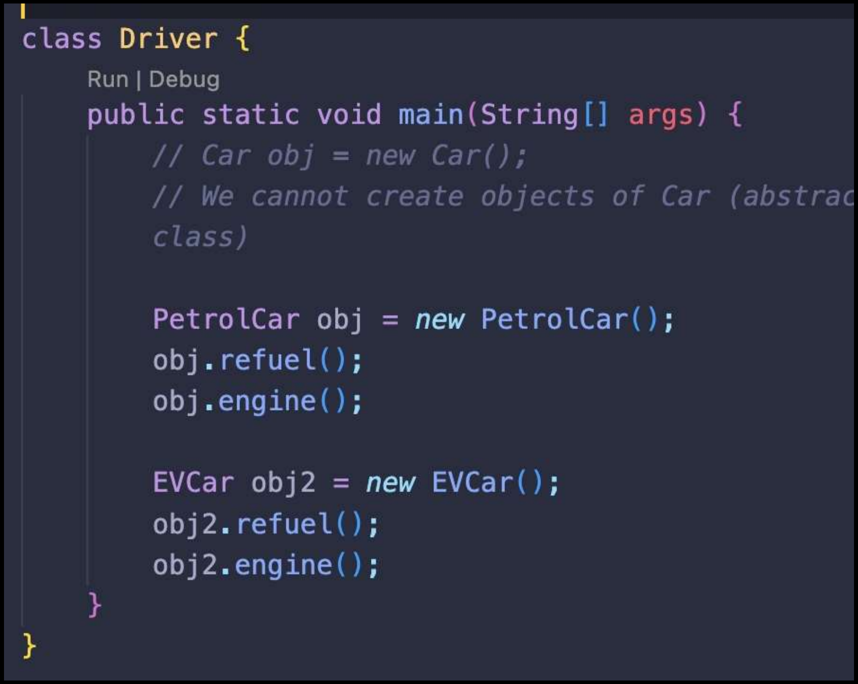
Specialized class

abstract refuel, engine

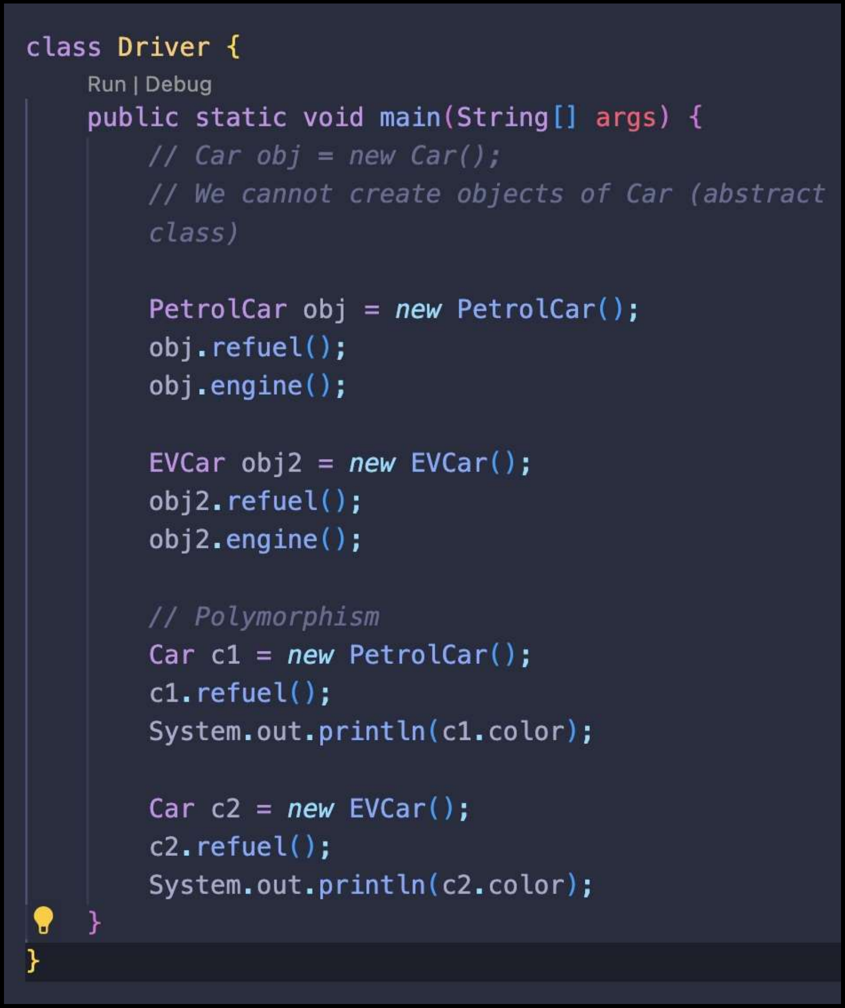
$$\begin{tikzcd} & \text{Car} & \\ \text{PetrolCar} \arrow[ur, "\text{extends}"] & & \text{EVCar} \arrow[ul, "\text{extends}"'] \end{tikzcd}$$

@override refuel, engine

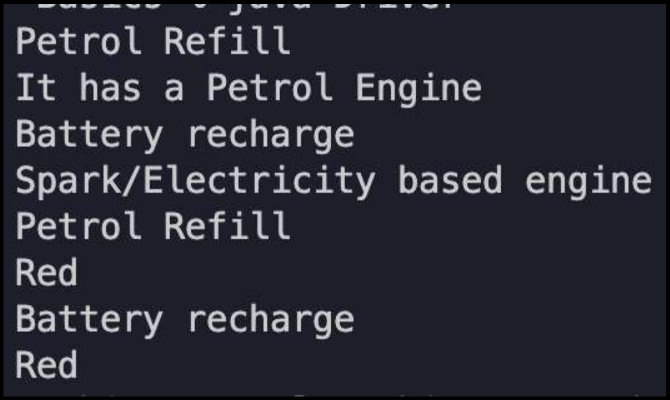
@override refuel, engine



```java
class Driver {
    Run | Debug
    public static void main(String[] args) {
        // Car obj = new Car();
        // We cannot create objects of Car (abstrac
        class)

        PetrolCar obj = new PetrolCar();
        obj.refuel();
        obj.engine();

        EVCar obj2 = new EVCar();
        obj2.refuel();
        obj2.engine();
    }
}
```



```java
class Driver {
    Run | Debug
    public static void main(String[] args) {
        // Car obj = new Car();
        // We cannot create objects of Car (abstract
        class)

        PetrolCar obj = new PetrolCar();
        obj.refuel();
        obj.engine();

        EVCar obj2 = new EVCar();
        obj2.refuel();
        obj2.engine();

        // Polymorphism
        Car c1 = new PetrolCar();
        c1.refuel();
        System.out.println(c1.color);

        Car c2 = new EVCar();
        c2.refuel();
        System.out.println(c2.color);
    }
}
```



```
Petrol Refill
It has a Petrol Engine
Battery recharge
Spark/Electricity based engine
Petrol Refill
Red
Battery recharge
Red
```

Q) Can abstract class have as abstract? (a) zero (b) some (c) all methods

- (a) zero ⇓ yes ↓ all concrete methods
- (b) some ⇓ yes (0% – 100%)
- (c) all methods ⇓ yes ↓ all abstract methods

Q) Can there be an abstract method inside a concrete (non-abstract) class?

NO, concrete class will throw compilation error
abstract method can be defined in abstract class only.

Q) Can abstract methods be (a) final (b) static (c) private?

↓ overriding ✓

- (a) final ↓ Overriding ✗ — no
- (b) static ↓ method hiding ✓, overriding ✗, early binded — no
- (c) private ↓ early binded ✓, overriding ✗ — no

Q) Constructors & abstract classes :-

(a) can there be constructors inside a abstract class? Yes

{to instantiate child class objects}

↳ constructor chaining super();

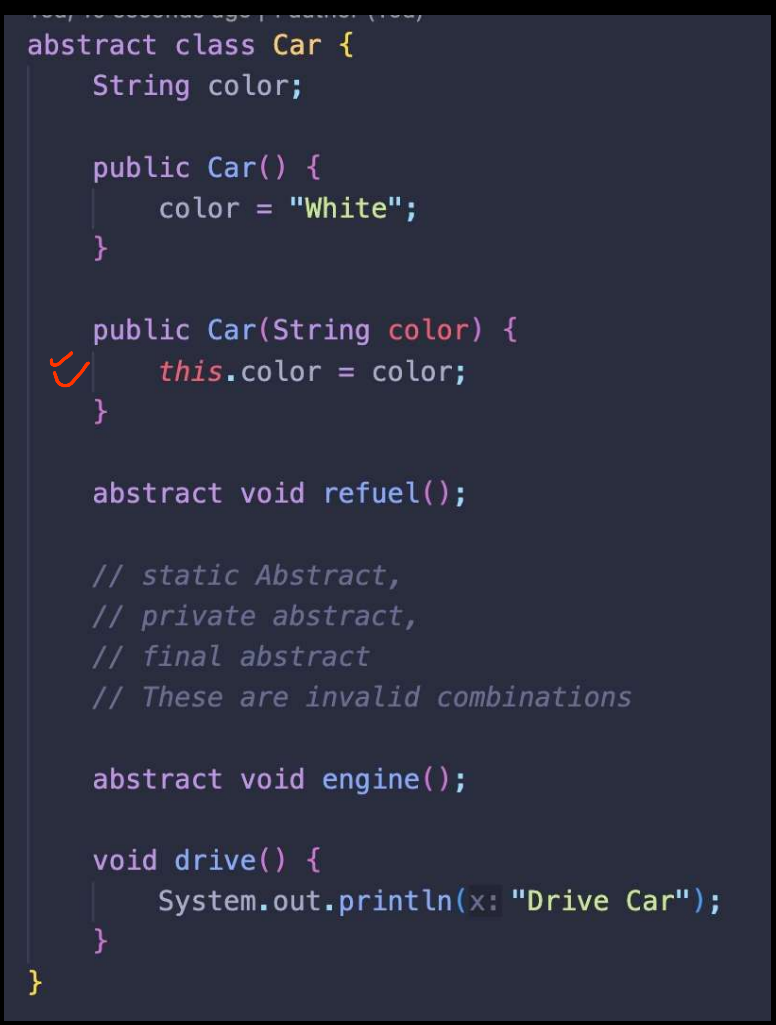
(b) can constructor itself be abstract? abstract constructors no

overriding ✗ — overriding ✓

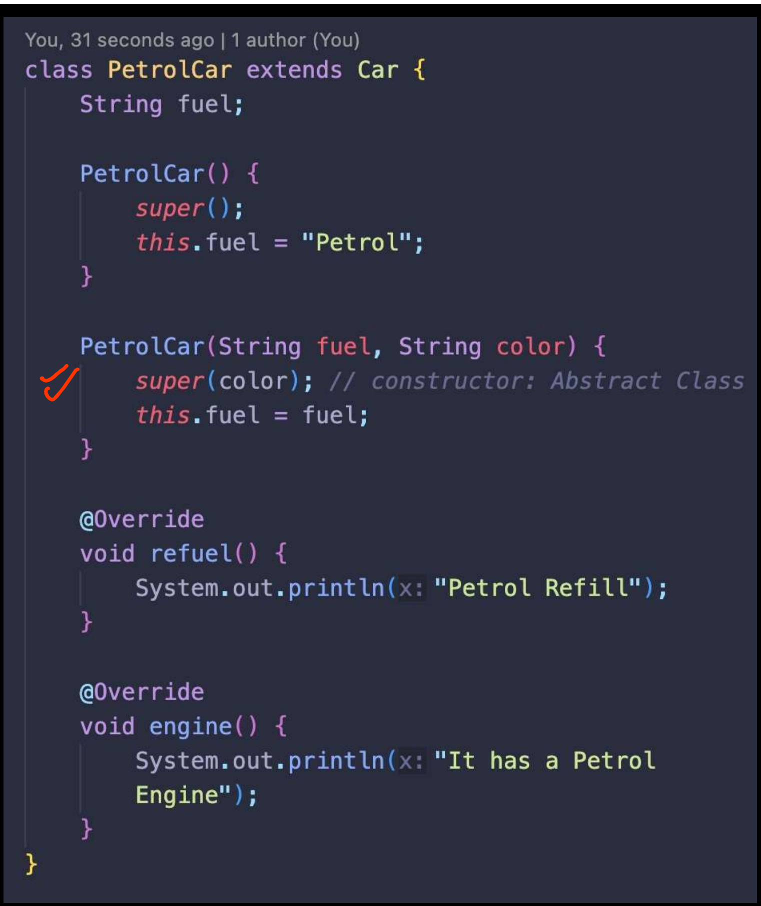
(c) Does abstract class have "this" keyword?

↳ yes → due to object creation of child

en

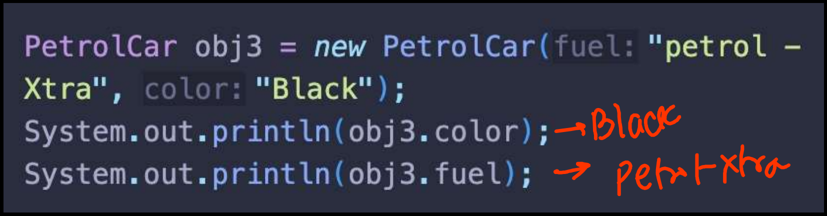


```java
abstract class Car {
    String color;

    public Car() {
        color = "White";
    }

    public Car(String color) {
✓       this.color = color;
    }

    abstract void refuel();

    // static Abstract,
    // private abstract,
    // final abstract
    // These are invalid combinations

    abstract void engine();

    void drive() {
        System.out.println(x: "Drive Car");
    }
}
```



You, 31 seconds ago | 1 author (You)

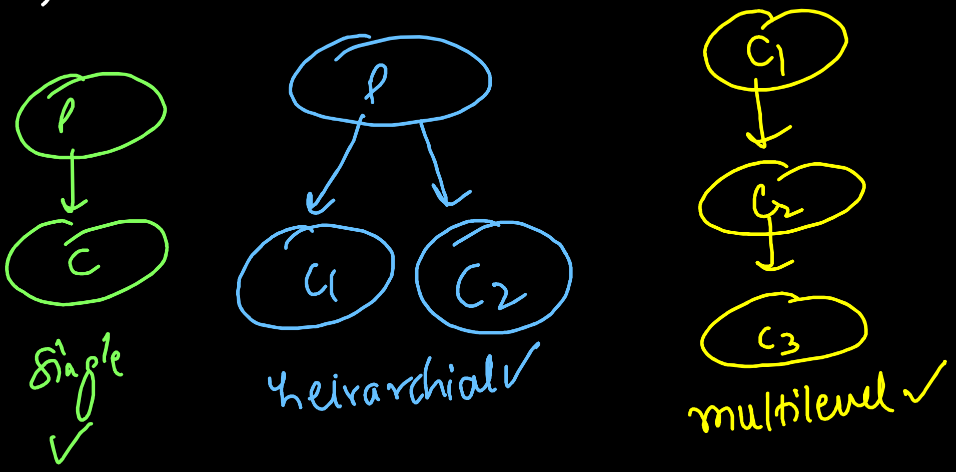
```java
class PetrolCar extends Car {
    String fuel;

    PetrolCar() {
        super();
        this.fuel = "Petrol";
    }

    PetrolCar(String fuel, String color) {
✓       super(color); // constructor: Abstract Class
        this.fuel = fuel;
    }

    @Override
    void refuel() {
        System.out.println(x: "Petrol Refill");
    }

    @Override
    void engine() {
        System.out.println(x: "It has a Petrol
        Engine");
    }
}
```

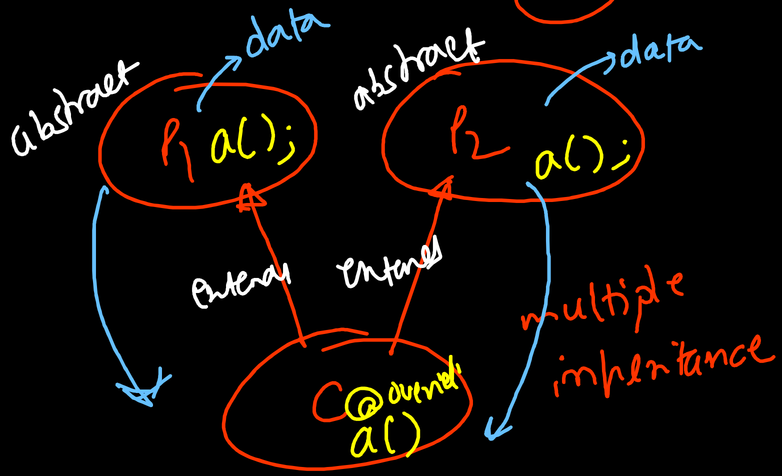


```java
PetrolCar obj3 = new PetrolCar(fuel: "petrol -
Xtra", color: "Black");
System.out.println(obj3.color); → Black
System.out.println(obj3.fuel); → petrol-Xtra
```

Q) Can we implement multiple inheritance using abstract classes?

↳ no





data X 2

↳ ambiguity

multiple inheritance not allowed

Q) What is the difference between encapsulation, data hiding and data abstraction?

Encapsulation :→ Wrapping together properties & behavior in a single entity known as class.

Data Hiding :→ Properties / Behaviors $\longrightarrow$ access modifiers

public

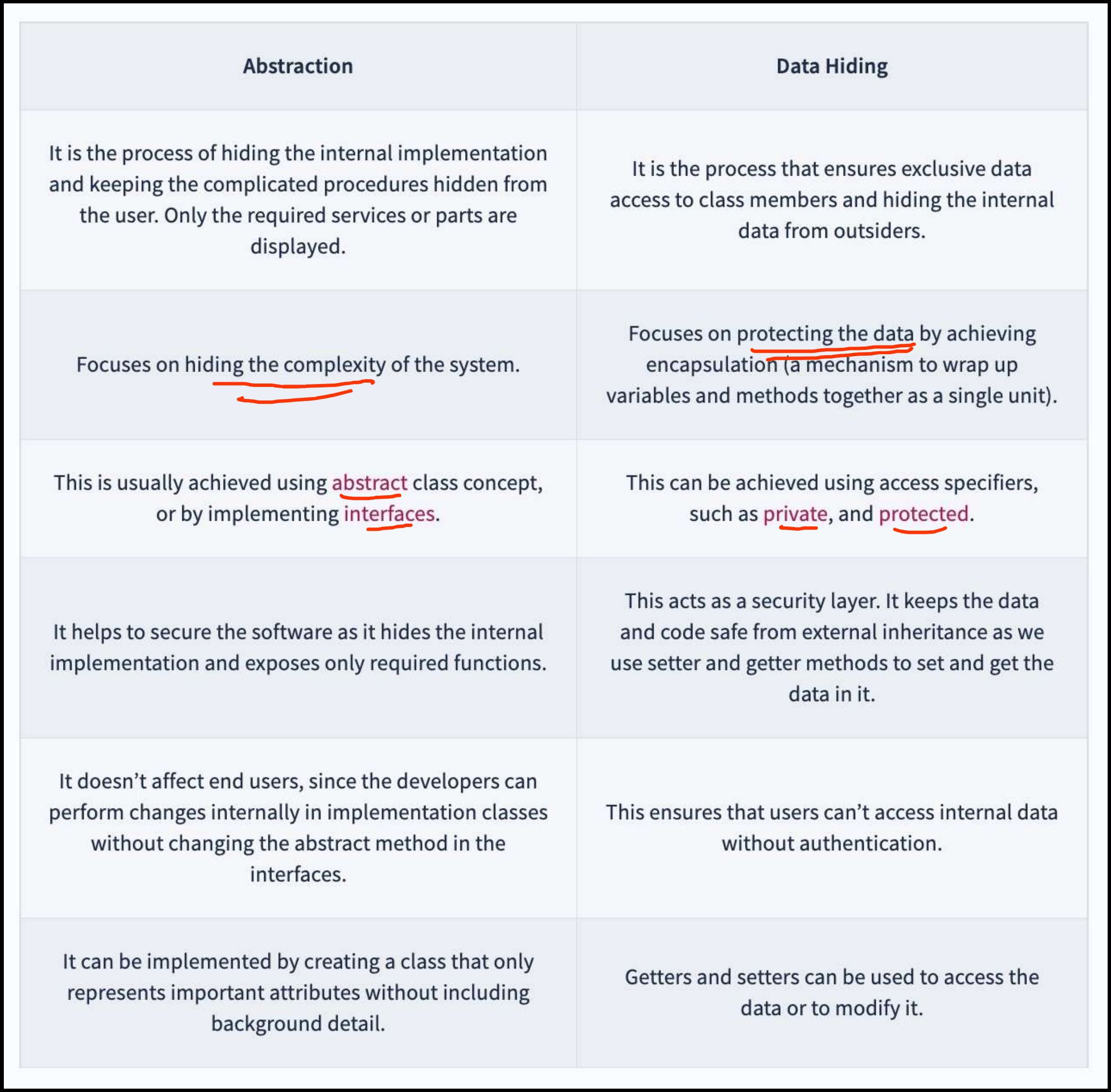
default/protected

private

Abstraction :→ Hiding the unneccessary details from the client

abstract class

interface



Abstraction	Data Hiding
It is the process of hiding the internal implementation and keeping the complicated procedures hidden from the user. Only the required services or parts are displayed.	It is the process that ensures exclusive data access to class members and hiding the internal data from outsiders.
Focuses on hiding the complexity of the system.	Focuses on protecting the data by achieving encapsulation (a mechanism to wrap up variables and methods together as a single unit).
This is usually achieved using abstract class concept, or by implementing interfaces.	This can be achieved using access specifiers, such as private, and protected.
It helps to secure the software as it hides the internal implementation and exposes only required functions.	This acts as a security layer. It keeps the data and code safe from external inheritance as we use setter and getter methods to set and get the data in it.
It doesn't affect end users, since the developers can perform changes internally in implementation classes without changing the abstract method in the interfaces.	This ensures that users can't access internal data without authentication.
It can be implemented by creating a class that only represents important attributes without including background detail.	Getters and setters can be used to access the data or to modify it.