JavaScript Code Example

```
class Car {
  // Private fields
  #engineStatus;

  // Public constructor
  constructor(brand, model, mileage) {
    this.brand = brand; // Public property
    this.model = model; // Public property
    this.mileage = mileage; // Public property
    this.#engineStatus = false; // Private property (engine is off by default)
  }

  // Public method to start the car
  startCar() {
    if (!this.#engineStatus) {
      this.#engineStatus = true;
      console.log(`${this.brand} ${this.model} has started.`);
    } else {
      console.log(`${this.brand} ${this.model} is already running.`);
    }
  }

  // Public method to stop the car
  stopCar() {
    if (this.#engineStatus) {
      this.#engineStatus = false;
      console.log(`${this.brand} ${this.model} has stopped.`);
    } else {
      console.log(`${this.brand} ${this.model} is already stopped.`);
    }
  }

  // Public method to get the mileage
  getMileage() {
    return `Mileage: ${this.mileage} miles.`;
  }

  // Private method (can only be accessed inside the class)
  #isEngineRunning() {
    return this.#engineStatus;
  }

  // Public method to check car status
  checkStatus() {
    if (this.#isEngineRunning()) {
      console.log(`${this.brand} ${this.model} is running.`);
    } else {
      console.log(`${this.brand} ${this.model} is not running.`);
    }
  }
```

```
}

// Create an instance of the Car class
const myCar = new Car("Toyota", "Corolla", 50000);

// Public methods can be called directly
myCar.startCar(); // Output: Toyota Corolla has started.
myCar.checkStatus(); // Output: Toyota Corolla is running.

myCar.stopCar(); // Output: Toyota Corolla has stopped.
myCar.checkStatus(); // Output: Toyota Corolla is not running.

console.log(myCar.getMileage()); // Output: Mileage: 50000 miles.

// Trying to access private fields or methods will throw an error
console.log(myCar.#engineStatus); // SyntaxError: Private field '#engineStatus'
must be declared in an enclosing class
console.log(myCar.#isEngineRunning()); // SyntaxError: Private field
'#isEngineRunning' must be declared in an enclosing class
```

Explanation

1. **Private Fields and Methods**:
   - Private properties like `#engineStatus` and private methods like `#isEngineRunning()` are only accessible within the class. This prevents direct modification or access from outside the class, ensuring the internal state remains secure and is manipulated only in controlled ways.
   - For example, `#engineStatus` is used internally to check whether the car engine is running. Outside access is restricted to ensure this critical data isn't altered accidentally.

2. **Public Methods and Properties**:
   - Public properties (`brand`, `model`, `mileage`) and methods like `startCar()`, `stopCar()`, `checkStatus()`, and `getMileage()` allow external code to interact with the object safely. They expose a controlled interface for modifying the object's internal state or retrieving information without allowing direct manipulation of sensitive properties.
   - For instance, the `startCar()` and `stopCar()` methods provide a safe way to change the car's engine status, with conditions in place to avoid starting an already running car or stopping a car that's already off.

Why Use Private and Public Properties and Methods?

1. **Encapsulation**:
   - Private fields keep sensitive data hidden, preventing it from being modified directly by external code. This helps avoid unintended behavior caused by unauthorized changes to key properties.

2. **Controlled Access**:

- Public methods control how external code interacts with private data. For example, the car can only be started or stopped via `startCar()` or `stopCar()`, ensuring conditions like "car already running" are handled internally.

3. **Security and Maintainability**:
   - Private fields ensure that important data (like engine status) cannot be accessed or modified in unexpected ways. The logic within the class can be refactored or updated without breaking the way external code interacts with the object.

---

1. **Why can't private fields be accessed directly outside the class?**

- **Answer**: Private fields (denoted by `#`) are part of the internal implementation of the class, and by design, they cannot be accessed outside of the class definition. This ensures encapsulation, preventing external code from modifying or retrieving sensitive internal data directly.

2. **Can private fields be inherited by subclasses?**

- **Answer**: No, private fields cannot be inherited by subclasses. Private fields are only accessible within the class where they are defined, not in subclasses or external instances. If a subclass needs access to the data, you can expose it through public getter or setter methods.

3. **What happens if you try to assign a new value to a private field from outside the class?**

- **Answer**: Trying to access or modify a private field from outside the class will result in a `SyntaxError`. Private fields can only be read or written within the class, using methods that are part of that class.

```
const myCar = new Car("Toyota", "Corolla", 50000);
myCar.#engineStatus = true; // SyntaxError: Private field '#engineStatus' must be declared in an enclosing class
```

4. **How can a private field be updated indirectly without exposing the field itself?**

- **Answer**: You can provide **public setter methods** that control how the private field is updated. This allows external code to interact with the private data safely without exposing it directly.

```
class BankAccount {
  #balance;

  constructor(balance) {
    this.#balance = balance;
  }

  deposit(amount) {
    if (amount > 0) {
      this.#balance += amount; // Indirectly updating private field
    }
```

```
  }

  getBalance() {
    return this.#balance; // Indirectly accessing private field
  }
}

const account = new BankAccount(100);
account.deposit(50); // OK
console.log(account.getBalance()); // Outputs: 150
console.log(account.#balance); // SyntaxError
```

5. **Is it possible to have both public and private methods with the same name in a class?**

- **Answer**: No, it's not possible to have both a public and a private method with the same name in the same class. JavaScript will treat them as conflicting declarations, and the code will throw an error.

6. **What happens if you define a public method with the same name as a private method in a subclass?**

- **Answer**: Since private methods are not visible outside the class in which they are defined, defining a public method with the same name in a subclass does not cause a conflict. The subclass method is treated as a separate, independent method.

```
class Parent {
  #privateMethod() {
    console.log("Private method in Parent class");
  }
}

class Child extends Parent {
  #privateMethod() {
    console.log("Private method in Child class");
  }
}

const child = new Child();
// This will call the child's private method without any conflict, as both are
separate
```

7. **Can a public method access a private field directly? Why or why not?**

- **Answer**: Yes, a public method defined inside the same class can access a private field directly. This is one of the main purposes of private fields: they can be controlled through public methods, ensuring that only the class's internal methods can read or modify private fields.

```
class Example {
  #privateField = "private";
```

```
  showPrivateField() {
    console.log(this.#privateField); // Can access private field within public
method
  }
}

const obj = new Example();
obj.showPrivateField(); // Output: private
```

Tricky Test Scenarios

1. **What happens if you mistakenly try to create a private property without the # prefix?**
   - **Test**: Write a class with a "private" property, but forget to add # in front.

```
class MyClass {
  privateField = "secret"; // No '#' - This is not truly private
}

const obj = new MyClass();
console.log(obj.privateField); // What will happen here?
```

   - **Answer**: The property will not be private. It will be treated as a regular public property, and you will be able to access it directly from outside the class, which breaks encapsulation.

---

2. **Can private methods or fields be dynamically created or modified from outside the class?**
   - **Test**: Try to dynamically add a private field after creating an object.

```
class Person {
  constructor(name) {
    this.name = name;
  }
}

const john = new Person("John");
john.#privateData = "Cannot add"; // SyntaxError
```

   - **Answer**: No, private fields and methods cannot be dynamically added or modified from outside the class. Attempting to do so will result in a `SyntaxError` since private fields are restricted to the class where they are declared.

---

3. **What happens if you define a getter and setter for a private field, and then directly try to modify the private field?**

- **Test**: Create a class with a private field and a public getter and setter.

```
class Rectangle {
  #width;

  constructor(width) {
    this.#width = width;
  }

  get width() {
    return this.#width;
  }

  set width(value) {
    if (value > 0) {
      this.#width = value;
    }
  }
}

const rect = new Rectangle(10);
rect.width = 20; // Valid update through setter
console.log(rect.width); // Outputs: 20
rect.#width = 30; // What happens here?
```

- **Answer**: The `rect.width` getter and setter will work as expected, but directly accessing `rect.#width` will result in a `SyntaxError` because `#width` is a private field and cannot be accessed outside the class.