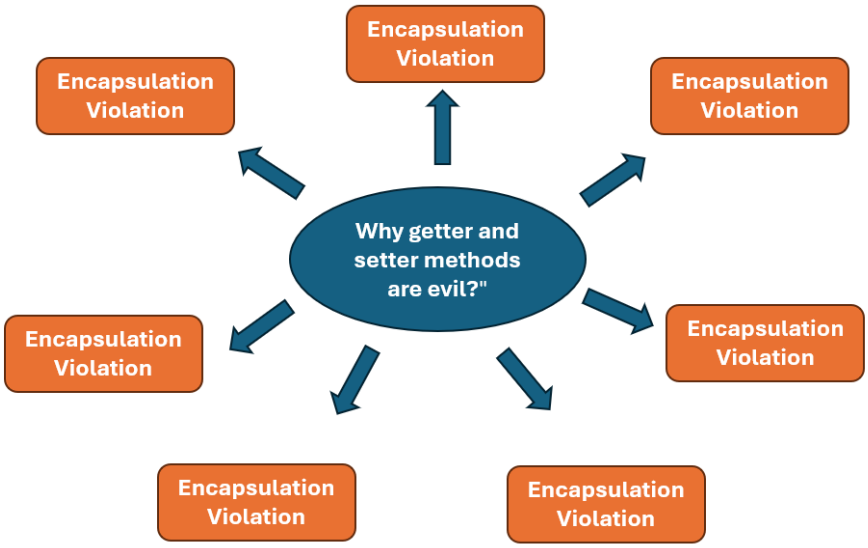
Based on Allen Holub's article "**Why getter and setter methods are evil?**" and supplemental research, here's a mind map summarizing the key points:

1. **Encapsulation Violation**
    - Getters and setters expose implementation details.
    - Encapsulation ensures that an object's data is hidden, supporting modularity.
2. **Object-Oriented Principles**
    - Getters/setters don't align well with object-oriented (OO) design.
    - OO encourages data abstraction and the "tell, don't ask" approach—i.e., ask objects to do work rather than accessing their data.
3. **Maintenance Issues**
    - Changes in data types require extensive code updates.
    - High maintainability is achieved when implementation changes minimally affect other code.
4. **Procedural Influence**
    - Getters/setters often stem from procedural programming habits.
    - Procedural thinking encourages direct data manipulation, which conflicts with OO design.
5. **JavaBeans Exception**
    - Getters/setters are used in JavaBeans for UI tools, but ideally, other interfaces handle these needs.
    - Direct calls to accessors for properties complicate code unnecessarily.
6. **Design Strategy (CRC Cards)**
    - Use CRC (Class-Responsibility-Collaborator) cards to define object roles and interactions.
    - Focus on message-passing between objects instead of data access.
7. **When Acceptable**
    - Some boundary layer classes (like database connectors) may require accessors due to flexible interface demands.
    - Encapsulated returns are acceptable if they're interfaces, keeping details hidden.


Encapsulation Violation
Encapsulation Violation
Encapsulation Violation
Why getter and setter methods are evil?"
Encapsulation Violation
Encapsulation Violation
Encapsulation Violation
Encapsulation Violation