**Takeaways**

Will be updated soon!

**Interview Corner**

* Why are public data members in a class a major security issue?
* Are accessor methods vulnerabilities in a class? Why or why not?
* Why is it important for a programmer to know the range of valid values for each data element being stored in the class?
* Is it acceptable to omit default constructors in a class? Why or why not?
* Why should the data types for class data members be chosen carefully?
* Why should a mutator method validate the input before changing the data values stored in an object?
* Why is it important for a programmer to double check any operations that change the value of class data members?

**🔐 1. Why are public data members in a class a major security issue?**

**Answer:**  
Public data members allow direct access and modification from outside the class, breaking the principles of encapsulation. This can lead to:

* Invalid data being assigned (e.g., negative age).
* Lack of control over how data is used or changed.
* Difficulty in debugging and maintaining code.
* Increased risk of unintended behavior.

Proper encapsulation using private fields and accessor/mutator methods ensures data integrity and security.

**🔎 2. Are accessor methods vulnerabilities in a class? Why or why not?**

**Answer:**  
No, accessor methods (getters) are **not vulnerabilities** by themselves. They **safely expose read-only access** to internal data without allowing modifications. However, overusing them or exposing sensitive internal structures (like returning a direct reference to a list or array) **can be risky**. Best practice is to:

* Return copies for mutable objects.
* Provide only necessary getters to reduce exposure.

**🎯 3. Why is it important for a programmer to know the range of valid values for each data element being stored in the class?**

**Answer:**  
Understanding valid value ranges helps ensure:

* **Data validation:** Prevents storing invalid or out-of-range values (e.g., age = -10).
* **Logical consistency:** Maintains correct program behavior.
* **Efficient error handling:** Detects and avoids bugs early.
* **Performance optimization:** Reduces unnecessary checks or memory use.

**🚫 4. Is it acceptable to omit default constructors in a class? Why or why not?**

**Answer:**  
Yes, it is acceptable *only if* you don’t need to create objects without passing parameters. However:

* If you define **any constructor with parameters**, Java does **not automatically provide** a default constructor.
* Omitting it can lead to errors if frameworks or tools expect one (e.g., serialization or reflection-based libraries).

✅ Best practice: include a default constructor if flexibility or framework compatibility is needed.

**🧠 5. Why should the data types for class data members be chosen carefully?**

**Answer:**  
Choosing appropriate data types:

* **Ensures memory efficiency** (e.g., using int instead of long if range fits).
* **Improves performance** by avoiding unnecessary overhead.
* **Avoids type errors** or data loss (e.g., float precision vs. double).
* **Clarifies intent** to others reading the code.
* **Helps in validation** (e.g., using enums instead of strings for limited options).

**✅ 6. Why should a mutator method validate the input before changing the data values stored in an object?**

**Answer:**  
Mutator methods (setters) must validate input to:

* **Prevent illegal or harmful values.**
* **Maintain internal consistency** (e.g., GPA must be 0.0–4.0).
* **Avoid runtime errors.**
* **Reduce debugging effort.**
* **Provide meaningful error messages** to users or other developers.

Without validation, objects can enter an invalid state, leading to unpredictable behavior.

**🔁 7. Why is it important for a programmer to double check any operations that change the value of class data members?**

**Answer:**  
Operations that change internal state should be verified because:

* Mistakes can corrupt the object’s state.
* They can introduce **bugs that are hard to detect** (especially in multi-threaded code).
* They might **violate business logic** or constraints.
* It ensures the object **remains valid and consistent** after any operation.

🛠️ Careful checks improve reliability and maintainability.