January 17, 2025

LEC 05 - Representation Invariants

Representation Invariants:

Key Idea: How do we document properties that must be true for every instance of a class at all times?

- Every instance attribute has a type annotation, which restricts the type of value the attribute can have (ex. `message: str`)
- We often want to apply more specific restrictions
    - Ex. Restricting tweets to 280 characters

Representation Invariant: A property of the instance attributed (including type annotation) that every instance of the class must satisfy

Takeaways:

1. Why should we care about representation invariants
    - We want user to know what we are doing
2. How do we enforce representation invariants
    - Assume they are true

Preconditions:

- `self` is an instance of the class, so all of the classes representation invariants are satisfied when the method is called
- The representation invariants of the class are preconditions for self and all class methods

---

Enforcing Representation Invariants:

- Every method must ensure that `self` satisfies all representation invariants after the method ends
    - Representation invariants are post conditions of `self` for the method
- All strategies have their own pros and cons

Strategy 1 - Preconditions:

- Require client code to be called with “good inputs” so that the method won’t violate the representation invariants

Strategy 2 - Ignoring Bad Inputs:

- Accept a wide range of inputs, and if an input would cause a representation invaraints to be violated, do nothing instead
- Also known as failing silently

January 17, 2025

LEC 05 - Representation Invariants

Strategy 3 - Fix Bad Inputs:

- Accept a wide range of inputs, and if an input would cause a representation invariant to be violated, change it to a “reasonable” or default value before continuing with the rest of the function

---

Direct Attribute Access:

- Even if our methods work properly, client code can access and mutate most instance attributes directly
- Documenting representation invariants is essential so that the client can ensure the attributes meet the requirements

---

More Design Considerations:

- When adding a new feature in a class, consider what you already have and what you can’t implement without extra attributes / methods
- Redundant information is bad
    - Memory space inefficiency
    - Prone to bugs