L09 - Abstract Data Types

Abstraction

- Omit/hide lower level details with a simple higher-level idea
- Encapsulation: Build components in a way that shields them from bugs in other parts of hte program. Modules are responsible for managing their internal state/behaviour
- Separation of Concerns: Make each feature the responsibility of a singular module, prevent too much functionality per module
- Information Concealing: Hide lower-level details not in specification, leaving user with only required information for implementation

Modularity

- Use small componenets to design larger system, with each component being self-contained, implementable, reusable, testable on its own

Abstract Types

"An abstract data type is characterized by operations that can be performed on it".

- Characterized by possible operations on the type rather than how it stores values
- Users create their own types.

Constructor: Create new objects of a type. $t* \rightarrow t$

Producer: Create new objects from old objects of the same type. ie: `concat()` creates new string from two existing ones. $T*, t* \rightarrow t$

Mutator: Alter object. $T+, t* \rightarrow$ void

Observer: Return an attribute of an abstract type. ie: `size()`. $T+, t* \rightarrow t$

Rules of Abstract Types

- Few, simple operations that can be combined in poewrful ways > many complex operations.
- Each operation should have well-defined purpose instead of many special cases.
- Set of operatoins should live up to client expectations, convey information clients want.
- Generic types should contain no domain-specific operations. ie. If storing a generic list, do not have operations that only work on a list of numbers.
- If type has a specific domain, do not have too many generic operations

Choosing Representation

Internal data structure to support operations on an abstract data type. Collection of fields in a class.

Usage of abstract type should be independent from its implementation.

- Changes in representation should not impact any client code
- Requires all operatoins to have preconditions, postconditions, frame conditions

Representation exposure is bad

- Keep representation inaccessible outside of the class. (`private`, `final`)

Preserving Invariants

Good ADT must preserve its own invariants, prevent clients from breaking them

Invariant - Property of an object that is always true

Preserving invariants mak ???? **fill this in after he posts the notes ig**

Activity:

```
public class CharSet {
    private final BitSet bitSet;

   public CharSet() {
        this.bitSet = new BitSet();
    }

    public void insert (Character c) {
        bitSet.set((integer)c, 1);
    }

    public void delete(Character c) {
        bitSet.set((integer) c, 0);
    }

    public boolean member(Character c) {
        return bitSet.get((integer) c);
    }

    public int size() {
    int count = 0;
        for (int i : bitSet) {
          if (i == 1) { count++; }
    }
    return count
    }

}
```