ABSTRACT DATA TYPE

(ADT)

DEFINITIONS:

Abstract data types are an instance of a general principle in software engineering, which goes by many names with slightly different shades of meaning. Here are some of the names that are used for this idea:

- Abstraction. Omitting or hiding low-level details with a simpler, higher-level idea.
- Modularity. Dividing a system into components or modules, each of which can be designed, implemented, tested, reasoned about, and reused separately from the rest of the system.
- Encapsulation. Building walls around a module (a hard shell or capsule) so that the module is responsible for its own internal behavior, and bugs in other parts of the system can't damage its integrity.
- Information hiding. Hiding details of a module's implementation from the rest of the system, so that those details can be changed later without changing the rest of the system.
- Separation of concerns. Making a feature (or "concern") the responsibility of a single module, rather than spreading it across multiple modules.

COMMON OPERATIONS OVER AN ADT:

The operations of an abstract type are classified as follows:

- Creators create new objects of the type. A creator may take an object as an argument, but not an object of the type being constructed.
- Producers create new objects from old objects of the type. The concat method of String, for example, is a producer: it takes two strings and produces a new one representing their concatenation.
- Observers take objects of the abstract type and return objects of a different type. The size method of List, for example, returns an int.
- Mutators change objects. The add method of List, for example, mutates a list by adding an element to the end.

INVARIANTS

An ADT preserves its own invariants.

An *invariant* is a property of a program that is always true, for every possible runtime state of the program.

Immutability is one crucial invariant: once created, an immutable object should always represent the same value, for its entire lifetime. Saying that the ADT *preserves its own invariants* means that the ADT is responsible for ensuring that its own invariants hold. It doesn't depend on good behavior from its clients.

When an ADT preserves its own invariants, reasoning about the code becomes much easier. If you can count on the fact that Strings never change, you can rule out that possibility when you're debugging code that uses Strings – or when you're trying to establish an invariant for another ADT that uses Strings. Contrast that with a string type that guarantees that it will be immutable only if its clients promise not to change it. Then you'd have to check all the places in the code where the string might be used.

http://web.mit.edu/6.005/www/fa14/classes/08-abstract-data-types/