CSE 230: Data Structures

Lecture 3: Abstract Data Types (ADTs)

Dr. Vidhya Balasubramanian

Abstract Data Types

- Two important aspects of a data structure
  - Interface: describes what a data structure does
    - Also known as Abstract Data Type
    - Provides list of supported operations
    - Includes specification of types of arguments accepted and the return values
  - Implementation: describes how the data structure does it
    - Internal representation of data structure
    - Algorithms to implement the functionalities

Definition

- Abstract Data Type
  - Is a mathematical model for a certain class of data structures that have similar behavior; or for certain data types of one or more programming languages that have similar semantics [2]
  - e.g a stack ADT is defined by its operations, push, pop and top.
- ADT defines
  - the set of operations supported by a data structure and
  - the semantics, or meaning, of those operations

Implementation Strategy

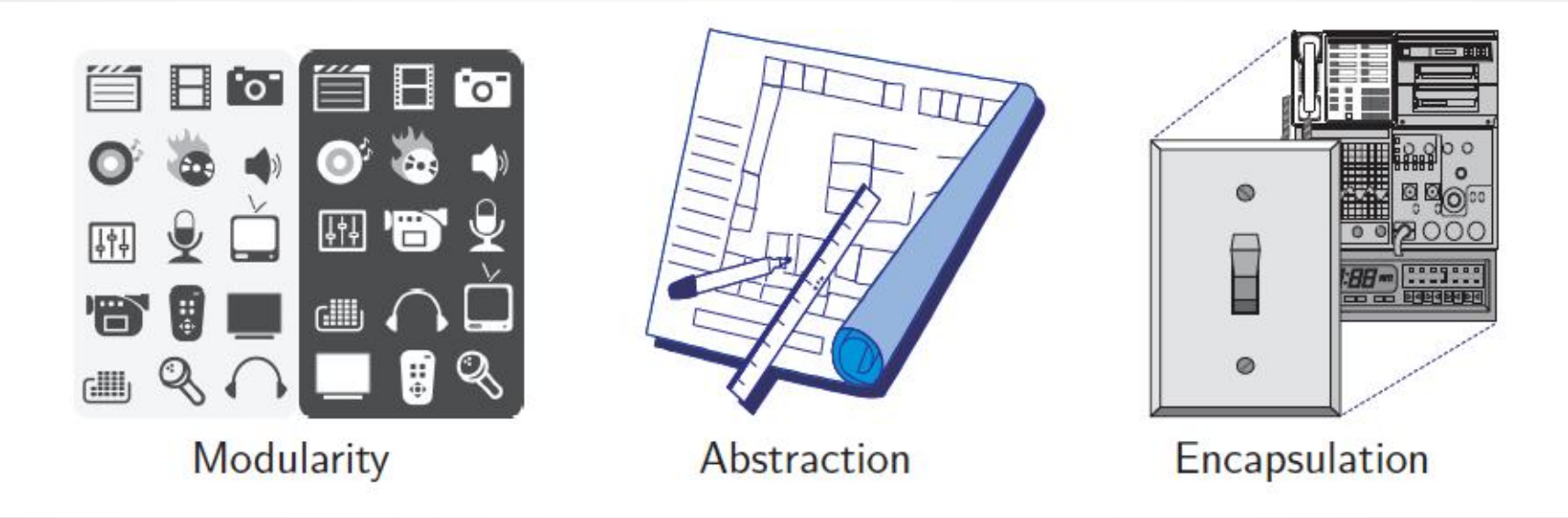
- Usually implemented as modules
  - the module's interface declares procedures that correspond to the ADT operations
- Can be multiple implementations of a single ADT
- Stack ADT Interface
  - Push(x)
  - Pop()
  - Top()

ADTs in Detail

- An abstract data type is defined as a mathematical model of the data objects that make up a data type as well as the functions that operate on these objects
- The definition can vary based on the type of programming language
  - Imperative
  - Object Oriented

Object Oriented Design Principles

- Modularity
- Abstraction
- Encapsulation



Src: 2014 Goodrich, Tamassia, Goldwasser

Advantages of Abstract Data Typing

- Encapsulation
  - Guarantees that the ADT has certain properties and abilities
  - User does not need to know the implementation details to use the ADT
- Localization of Change
  - Change to ADT implementation does not impact the code that uses the ADT
  - The implementation must still comply with the ADT definition, hence the interface will not change

Advantages

- Flexibility
  - Can use different implementations of an ADT in a code
    - All have same properties and abilities
  - The efficiency of different implementations maybe different
  - Can use the most suitable implementation

Interfaces and Classes

- The main structural element in Java that enforces an application programming interface (API) is an interface.
  - An interface is a collection of method declarations with no data and no bodies.
  - Interfaces do not have constructors and they cannot be directly instantiated.
- When a class implements an interface, it must implement all of the methods declared in the interface.

ADT using Object Orientation

- The ADT is defined as having variables and functions as defined in classes
- Interface defines the ADT (in Java)
  - Defines the functions and its parameters
- Class implements the ADT
  - Defines the variables
  - Implements the functions defined in the interface
  - Defines Constructors

ADT in Java

- While the functions of the data structures are same, the implementation of these functions can vary.
  - Use interfaces for defining the data structure i.e., defining the ADT
    - interface ISet {

```
void addElt(String newElt) ;
int remElt(String newElt) ;
int size();
boolean hasElt(String elt);
}
```

ADT in Java

- The class implements the interface
  - Implementation can depend on requirements
  - class Set implements ISet {

```
<Fields to hold the data>
void Set(String curElt){...}
void addElt(String newElt) { ... }
int remElt(String newElt) { ... }
int size() { ... }
boolean hasElt(String elt) { ... }
}
```