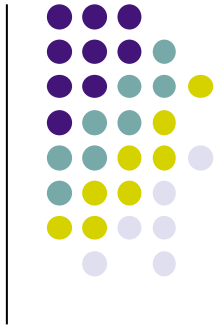
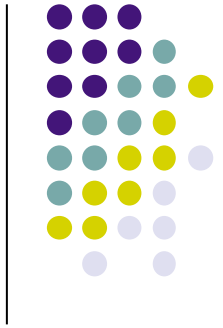
Data abstraction



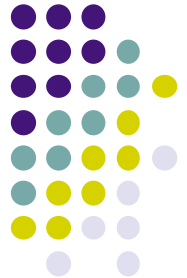
- Data abstraction is the main organizing principle for building complex software systems
  - Without data abstraction, computing technology would stop dead in its tracks
- We will study what data abstraction is and how it is supported by the programming language
  - The first step toward data abstraction is called encapsulation
  - Data abstraction is supported by language concepts such as higher-order programming, static scoping, and explicit state

Encapsulation



- The first step toward data abstraction, which is the basic organizing principle for large programs, is encapsulation
- Assume your television set is not enclosed in a box
  - All the interior circuitry is exposed to the outside
  - It's lighter and takes up less space, so it's good, right? NO!
- It's dangerous for you: if you touch the circuitry, you can get an electric shock
- It's bad for the television set: if you spill a cup of coffee inside it, you can provoke a short-circuit
  - If you like electronics, you may be tempted to tweak the insides, to "improve" the television's performance
- So it can be a good idea to put the television in an enclosing box
  - A box that protects the television against damage and that only authorizes proper interaction (on/off, channel selection, volume)

Encapsulation in a program

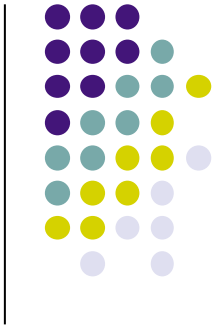


- Assume your program uses a stack with the following implementation:

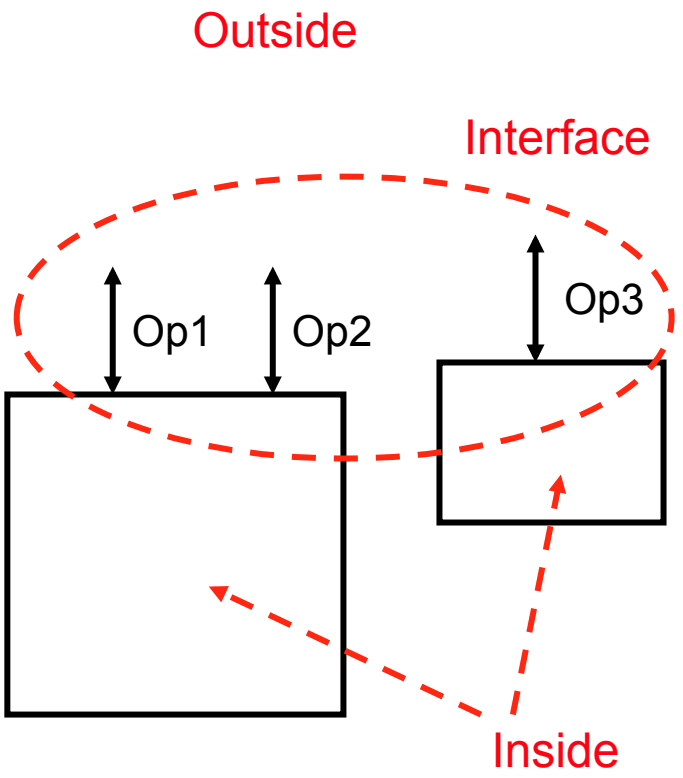
```
fun {NewStack} nil end
fun {Push S X} X|S end
fun {Pop S X} X=S.1 S.2 end
fun {IsEmpty S} S==nil end
```

- This implementation is not encapsulated!
  - It has the same problems as a television set without enclosure
  - It is implemented using lists that are not protected
    - A user can read stack values without the implementation knowing
    - A user can create stack values outside of the implementation
- There is no way to guarantee that an unencapsulated stack will work correctly
  - The stack must be encapsulated → data abstraction

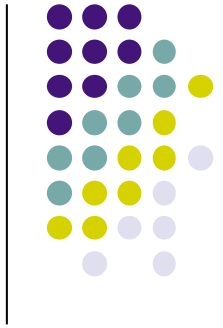
Definition of data abstraction



- A data abstraction is a part of a program that has an inside, an outside, and an interface in between
- The inside is hidden from the outside
  - All operations on the inside must pass through the interface, i.e., the data abstraction must use encapsulation
- The interface is a set of operations that can be used according to certain rules
  - Correct use of the rules guarantees that the results are correct
- The encapsulation must be supported by the programming language
  - We will see how the language can support encapsulation, that is, how it can enforce the separation between inside and outside

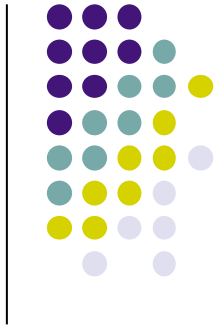


Advantages of data abstraction



- A guarantee that the abstraction will work correctly
  - The interface only allows well-defined interaction with the inside
- A reduction of complexity
  - The user does not have to know the implementation, but only the interface, which is generally much simpler
  - A program can be partitioned into many independent abstractions, which greatly simplifies use
- The development of large programs becomes possible
  - Each abstraction has a responsible developer: the person who implements it, maintains it, and guarantees its behavior
  - Each responsible developer only has to know the interfaces of the abstractions used by the abstraction
  - It's possible for teams of developers to develop large programs

The two main kinds of data abstraction



- There are two main kinds of data abstraction, namely objects and abstract data types
  - An object groups together value and operations in a single entity
  - An abstract data type keeps values and operations separate
- Some real world examples
  - A television set is an object: it can be used directly through its interface (on/off, channel selection, volume control)
  - Coin-operated vending machines are abstract data types: the coins and products are the values and the operations are the vending machines
- We will look at both objects and ADTs
  - Each has its own advantages and disadvantages