

# Abstract Data Types

Lecture 17

**Modules**

(part 1)

Chapter 7 of HR book

# Data Abstraction

- **Data abstraction** is perhaps the most important technique for structuring programs.
- Provides an ***interface*** that serves as a **contract** between the ***client*** and the ***implementor*** of an abstract type.
  - The interface specifies what the client may rely on for its own work, and, simultaneously, what the implementor must provide to satisfy the contract.

# Data Abstraction 2

- The interface **isolates** the client from the implementor so that each may be developed in isolation from the other
  - *data hiding*
- In particular, one implementation may be **replaced** by another without affecting the behavior of the client, provided that the two implementations meet the same interface.


# ADT

- An **abstract data type** (ADT) is a type equipped with a set of operations for manipulating values of that type.
- ADT is implemented by providing a **representation** type for the values of the ADT and an **implementation** for the operations defined on values of the representation type.
- What makes an ADT abstract is that the representation type is **hidden** from clients of the ADT. Consequently, the only operations that may be performed on a value of the ADT are the exposed ones.
- This ensures that the representation may be changed without affecting the behavior of the client.

# ADT in F#

- In F# this can be achieved via the use of *signatures* and *modules*
  - sig files (**file.fsi**) specify the interface/API
  - module declarations (**file.fs**) represent the implementors side
- They are "matched" by the compiler, which compiles a library file (**file.dll**)
- Then, the dll is opened and used, possibly interactively

# Howto: using fsharpc/fsc

- Open a terminal (in Windows cmd, or better use git bash) and go to the directory containing your files
- For our working example: run
  - `fsc -a set.fsi listFS.fs` 
- This will produce a library file `listFS.dll`. To use it you can run F# interactive from the shell like that:
  - `fsi -r listFS.dll`, or
  - `#r "ListFs"` inside the editor
- Now you can open the module and use it in your script file

# Howto: VS and Mono

- "Open" a new project (**solution**) and choose F# library. Choose the name and location of the *dll* to be generated
- Go to Solution Explorer (**View >> visual design**), remove for hygiene reasons the *\*fs* and *\*fsx* that VS generates for you.
- "Add existing files" (right click) namely the *\*fsi* and *\*fs*. Move the *\*fsi* to be first. "Build" the project
- To use the *dll*, you need to reference it in your *\*.fsx* file
  - `#r @"directory\name.dll"`
- Note that the *dll* will be under *bin\debug* in the folder that VS builds for you.