Abstraction

Abstraction layers / levels in cs

Computer science commonly presents levels (layers) of abstraction;
each level represents a different model of the same information and processes

DT (in programming)

a particular type of information 🡐 classification
A type denotes a domain (a set of values) and operations on those values.

Concrete data types

= available data types, ready to use !
The available data types vary from one programming language to another, but there are some that usually exist in one form or another.

ADT

ADT (abstract data type): = domain (of values)
and operations (specification)

abstraction = a way of hiding the implementation details

- data abstraction: domain of values; no indication about representation
- procedural abstraction: operation specification; no indication of how achieved
  mathematical constraints on the effects (and possibly cost)

Implementation

for an ADT => (concrete) DT

- can be implemented in many ways and in many programming languages
- ADTs are often implemented as classes:
  class interface declares procedures that correspond to the ADT operations.
  ➔**Encapsulation**
  → **information hiding strategy**

Advantages

A major goal of software engineering: write reusable code!

Reduced complexity easy to learn and use

Abstraction provides a promise that any implementation of the ADT has certain properties and abilities.This is required in order to use the ADT (implementation).
The user does not need any technical knowledge of how the implementation works to use the ADT. In this way, the implementation may be complex but will be encapsulated in a simple interface when it is actually used.

Flexibility

Different implementations of an ADT are equivalent and may be used somewhat interchangeably in code that uses the ADT.

Localization of change

Code that uses an ADT object will not need to be edited if the implementation of the ADT is changed.

Efficiency

Different implementations of an ADT may be more efficient in different situations; it is possible to use each in the situation where they are preferable, thus increasing overall efficiency.

Data structure

Data structure: is a description of how data are organized.
For any data type, in a computer we will have a representation.

- DS describe a possible representation for a domain

Two ways to think about a DS

- **logic** DS:
  specification of how informations of different types are grouped together
  - elements that form the DS and their relations
  - based on other DT and using ***DS builders***
- **physical representation: the way data is stored in computer memory**

data structure classification (based on physical representation)

- static:
  during execution, it occupies a fixed zone of memory
- semi-static:
  the memory area where the elements of the structure are stored – is fixed
  elements can change their place
- dynamic:
  occupies a memory zone that changes during program execution,
  according to the needs of dimension change

ADT and DS

For an ADT	in programs, we have:
- domain:	➔ data structure (representation)
- operations:	➔ algorithms for operations

<u>Remark:</u>
For a given ADT we can consider more than one DS to implement it. Each data structure has specific benefits as well as its drawbacks.

ADT	DS
Fixed-sized	static
Variable-sized	dynamic

ADT, DS and levels of abstraction

In practice,
we can have different levels of logic specification of structure used for storing data
(similar with stepwise refinement for algorithms)

Example:

ADT domain	collection of elements of type TE
DS:	
refinement 0	list of elements of type TE
refinement 1	singly linked list of elements of type TE
refinement 2	singly linked list with dynamically allocated nodes containing elements of type TE

Data structure: the lowest level of logic data structure specification
<u>Convention</u>: DS - described by using basic types and DS builders
(see conventions)

<u>Remarks</u>
1. Refinements involves (permits) some levels of abstraction. For example, we can study the: "single linked list" regardless of the fact that it is dynamically allocated or it has a semi-static representation.
2. The fundamental building blocks for data structures are provided by languages. (arrays, records, pointers/references)

Why DS and ADT ?

Algorithm efficiency strongly depends on the chosen DS for the concrete problem.

Selecting the most appropriate data structure to store your application's data is extremely important. Your choice of data structure affects the operation and performance of your application -- sometimes with little consequence, sometimes dramatically.
http://java.sun.com/docs/books/performance/1st_edition/html/JPAlgorithms.fm.html

Understanding **how to choose** the **best** algorithm or **data structure** for a particular task is one of the keys to writing high-performance software.