Data Structures and Algorithms

Lecture 01: Abstract Data Type

UNIVERSITY of GREENWICH
Alliance with FPT Education

Pearson BTEC

Contents

- What is Data Abstraction? What is ADT
- Model for an Abstract Data Type
- ADT Example
- ADT vs Object-Oriented Programming

UNIVERSITY of GREENWICH
Alliance with FPT Education
Pearson BTEC

What is Data Abstraction?

- Concept of "**Abstraction**"
  - Allows us to consider the high-level characteristics of something without getting bogged down in the details
  - For example: *Method abstraction* in OOP like C++, we can use (pre-defined) methods without concern how they really works inside
- **Data Abstraction**
  - We know what a data type can do
  - How it is done is hidden

UNIVERSITY of GREENWICH
Alliance with FPT Education
Pearson BTEC

What is an Abstract Data Type?

- Abstract Data Type (ADT)
  - Defines a particular data structure in terms of data and operations
  - Offers an interface of the objects (instances of an ADT)
- An ADT consists of
  - Declaration of data
  - Declaration of operations
  - Encapsulation of data and operations : data is hidden from user and can be manipulated only by means of operations

ADT example: Floating point numbers

- You don't need to know how much about floating point arithmetic works to use `float`
  - Indeed, the details can vary depending on processor, even virtual coprocessor
  - But the compiler hides all the details from you--some numeric ADTs are built-in
  - All you need to know is the syntax and meaning of operators, +, -, *, /, etc.
- Hiding the details of implementation is called **encapsulation** (**data hiding**)

UNIVERSITY of GREENWICH
Alliance with FPT Education
Pearson BTEC

ADT = properties + operations

- An **ADT** describes a set of objects sharing the same properties and behaviors
  - The **properties** of an ADT are its **data** (representing the internal state of each object
    - `double d;` -- bits representing exponent & mantissa are its data or state
  - The **behaviors** of an ADT are its **operations** or **functions** (operations on each instance)
    - `sqrt(d) / 2;` //operators & functions are its behaviors
- Thus, an ADT couples its data and operations
  - OOP emphasizes **data abstraction**

Formal, language-independent ADTs

- An ADT is a formal description, not code; independent of any programming language
  - *Why is code independence a good idea?*
- Promotes **design by contract**
  - Specify responsibilities of suppliers and clients explicitly, so they can be enforced, if necessary

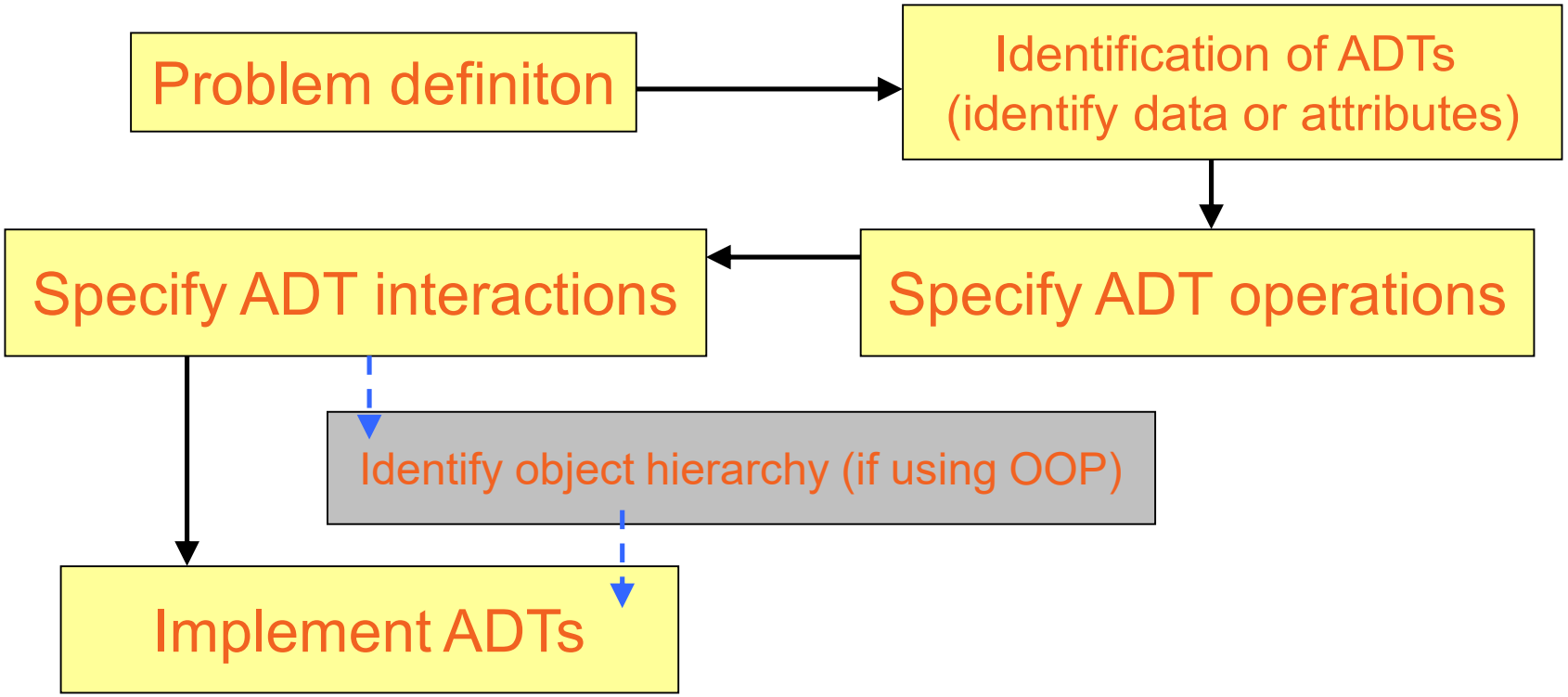
UNIVERSITY of GREENWICH
Alliance with FPT Education
Pearson BTEC

ADT Implementation

- Implementaion of an Abstract Data Type (ADT)
  - Hidden from the user
  - Same ADT may be implemented in different ways in different languages
  - Some languages offer built-in ADTs and/or features to be used to implement ADTs (user-define types)
- ADTs support modular design which is very important in software development

ADT from definition to implementation

System design with ADTs



UNIVERSITY of GREENWICH
Alliance with FPT Education

Pearson BTEC

Benefits

- Manufacturer (who creates ADT) benefits:
  - easy to modify, maintain
  - profitable
  - reusable
- Client (who uses ADT) benefits:
  - simple to use, understand
  - familiar
  - cheap
  - component–based

ADT example: LIST ADT

- A list contains elements of same type arranged in sequential order and following operations can be performed on the list
  - get() – Return an element from the list at any given position.
  - insert() – Insert an element at any position of the list.
  - remove() – Remove the first occurrence of any element from a non-empty list.
  - removeAt() – Remove the element at a specified location from a non-empty list.
  - replace() – Replace an element at any position by another element.
  - size() – Return the number of elements in the list.
  - isEmpty() – Return true if the list is empty, otherwise return false.
  - isFull() – Return true if the list is full, otherwise return false.

VDM Specification Language

- The Vienna Development Method (VDM) was originally developed at the IBM laboratories in Vienna in the 1970’
- A formal specification language intended to specify object oriented systems
- ADT can be specified by a formal language likes VDM.

ADT in VDM example: Boolean type

Name: Boolean

Symbol: `bool`

Values: `true`, `false`

Operators: Assume that `a` and `b` in the following denote arbitrary boolean expressions:

Operator	Name	Type
`not b`	Negation	`bool` → `bool`
`a and b`	Conjunction	`bool` * `bool` → `bool`
`a or b`	Disjunction	`bool` * `bool` → `bool`
`a => b`	Implication	`bool` * `bool` → `bool`
`a <=> b`	Biimplication	`bool` * `bool` → `bool`
`a = b`	Equality	`bool` * `bool` → `bool`
`a <> b`	Inequality	`bool` * `bool` → `bool`

UNIVERSITY of GREENWICH
Alliance with FPT Education
Pearson BTEC

ADT vs Object-Oriented Programming

- ADTs are not a part of a particular programming language
- Rather they are implemented by a programmer to solve a particular problem or some class of problems
- In OOP, an ADT can be easily modeled as a class.
  - An instance as an object
  - Data of ADT as properties or fields of a class
  - Operations as methods
- ADT ≠ OOP
- Classes in OOP offers more features than ADTs : Inheritance (Superclass-Subclass), Polymorphisms, etc.

UNIVERSITY of GREENWICH
Alliance with FPT Education
Pearson BTEC

ADT in Java: A Bag

- A bag is just a container for a group of data items
  - analogy: a bag of candy
- The positions of the data items don't matter (unlike a list)
  - {3, 2, 10, 6} is equivalent to {2, 3, 6, 10}
- The items do *not* need to be unique (unlike a set)
  - {7,2,10,7,5} isn't a set, but it is a bag

UNIVERSITY of GREENWICH
Alliance with FPT Education
Pearson BTEC

ADT in Java: A Bag

- The operations supported by our Bag ADT:
  - add(item): add item to the Bag
  - remove(item): remove one occurrence of item (if any) from the Bag
  - contains(item): check if item is in the Bag
  - numItems(): get the number of items in the Bag
  - grab(): get an item at random, without removing it
  - toArray(): get an array containing the current contents of the bag

ADT in Java: A Bag

- In Java, we can use interface to specify an ADT:

```
1 public interface Bag {
2     boolean add(Object item);
3     boolean remove(Object item);
4     boolean contains(Object item);
5     int numItems();
6     Object grab();
7     Object[] toArray();
8 }
```

UNIVERSITY of GREENWICH
Alliance with FPT Education

Pearson BTEC

ADT in Java: A Bag

- In Java, we can use implement an ADT by a class:

```
10  public class ArrayBag implements Bag {
11          private Object[] items;
12          private int numItems;
13          public boolean add(Object item) {
14              // code to add
15          }
16
17          ...
18  }
```

UNIVERSITY of GREENWICH
Alliance with FPT Education

Pearson BTEC

The End