Storing Data in Java

Variables

Primitive type (int, double, boolean, etc.)

- Variable name refers to a memory location containing a primitive value

Reference type (Object, String, Integer, MyClass, etc.)

- Variable name refers to a memory location containing a reference value for data belonging to an object

Data Structure

A *formal* organization of a set of data (e.g. variables)

e.g. Arrays: variables of a given type in an integer-indexed sequence

- int intArray[ ] = {1, 2}; int a = intArray[0]; intArray[1] = 5;

e.g. Objects: data member names used to index variables

- player.name, player.hits, player.team ... player.hits = 100;

Abstract Data Types (ADTs)

Purpose

Define interfaces for complex data structures

- Hide *(abstract)* implementation details of operations that query and update
- Operations defined independent of the element type (Java: “generic”)

Some Common ADTs

List: Sequence of elements. Elements may be inserted or removed from any position in the list

Stack: List with last-in, first-out (LIFO) behaviour (“most recent”)

e.g. call stack

Queue: List with first-in, first-out (FIFO) (“in-order”)

e.g. lining up at a fast-food restaurant or bank

Common ADTs, Cont'd

Tree: Graph with directed edges, each node has one parent (except root), no cycles.

e.g. Decision tree representing possible moves in a game of tic-tac-toe.

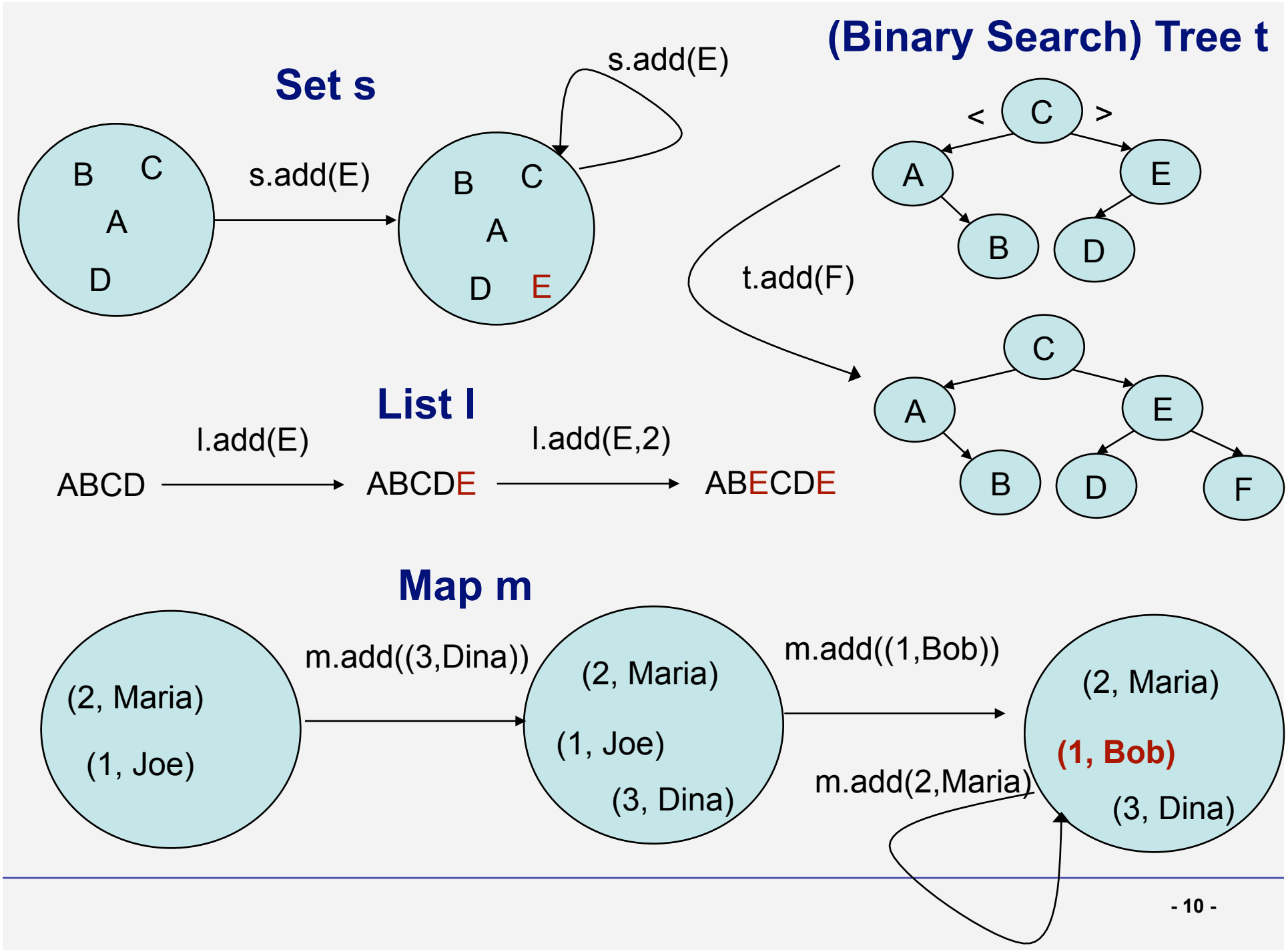
Set: Unordered group of unique items

e.g. Students in a class, the set of words in a text file

Map: Set of entries, each with unique key and (possibly non-unique) value

e.g. Student grade sheet: (StudentId, Grade)

e.g. Frequency of words in a text file (Word, Count)


Set s
s.add(E)
B C A D
s.add(E)
B C A D E
(Binary Search) Tree t
< C >
A E
B D
t.add(F)
C
A E
B D F
List l
l.add(E)
l.add(E,2)
ABCD
ABCDE
ABECDE
Map m
(2, Maria)
(1, Joe)
m.add((3,Dina))
(2, Maria)
(1, Joe)
(3, Dina)
m.add((1,Bob))
(2, Maria)
(1, Bob)
(3, Dina)
m.add(2,Maria)

Example: Implementing Abstract Data Types

List ADT

Represents series of elements, insertion and deletion of elements

Some Possible Implementations:

- Arrays:
  L.add(E) would copy E at the end of the array, L.get(4) returns 5th item in array
- Linked List: (objects forming a chain of references)
  L.add(E) would create a link from last node to a new node for E ; l.get(4) traverses the graph and then returns the 5th item

Choosing an Implementation for an ADT

Depending on common operations, some better than others

- Retrieving elements in list faster for array implementation
- Inserting, deleting elements faster for linked list implementation in general case

Ordering in ‘Unordered’ Sets and Maps

‘Unordered’ in Theory vs. in Code

In practice, an ordering of elements is used to implement a set, as memory and files store data as lists of bytes.

Sets

By definition, a set is an unordered group of unique elements

Maps

By definition, a map is a set of (key,value) pairs, where all keys are unique.

Ordering Sets and Maps

We can order the storage of set elements by:

1. The order in which elements are added (e.g. in a list)
2. The values of keys or data elements themselves (e.g. using a binary search tree)
3. A value computed for each element (“hash code”) that determines where an element is stored in a “hash table” (sophisticated ADT built on arrays); for maps, hash code computed using key value