# CS3500: Object-Oriented Design
# Fall 2013

## Class 7
## 9.27.2013



Northeastern University
College *of* Computer and Information Science

# What is data abstraction?

A type of abstraction that allows us to introduce new types of data objects.

# What must we define with a new data type?

- set of objects
- set of operations characterizing the behavior of the objects

```
data abstraction = <objects, operations>
```

# What do we gain from data abstraction?

"Data abstraction allows us to abstract from the details of how data objects are implemented to how the objects behave." (Liskov)

# Implementing Data Abstractions

- Representation

# Deriving a representation from an algebraic specification using Recipe

- Determine which operations of the ADT are basic creators, and which are other operations.
- Define an abstract class named T.
- For each basic creator c of the ADT, define a concrete subclass of T. For each concrete subclass, declare instance variables with the same types and names as the arguments that are passed to the corresponding basic creator c. For each concrete subclass, define a Java constructor that takes the same arguments as the basic creator c and stores those arguments into the instance variables.
- [So far we have defined the representation of T. Now wehave to define the operations.]

# Queue

- Similar to list
- First In, First Out (FIFO)

- Enqueue
- Dequeue

# Immutable Queue Algebraic Specs

```
public static methods:
empty  :              -> Queue
enqueue: Queue x int -> Queue
dequeue: Queue        -> Queue
first  : Queue        -> int
size   : Queue        -> int
isEmpty: Queue        -> boolean

Queue.dequeue(Queue.enqueue(q,i))  = q  if(Queue.isEmpty(q))
Queue.dequeue(Queue.enqueue(q,i))
   = Queue.enqueue(Queue.dequeue(q), i) if(!Queue.isEmpty(q))
Queue.first(Queue.enqueue(q,i)) = i                 if(Queue.isEmpty(q))
Queue.first(Queue.enqueue(q,i)) = Queue.first(q) if(!Queue.isEmpty(q))
Queue.size(Queue.empty()) = 0
Queue.size(Queue.enqueue(q,i)) = 1 + Queue.size(q)
Queue.isEmpty(Queue.empty()) = true
Queue.isEmpty(Queue.enqueue(q,i) = false
Queue.empty().toString() = ""
Queue.enqueue(q,i).toString() = q.toString() +"(" + i + ")"
```

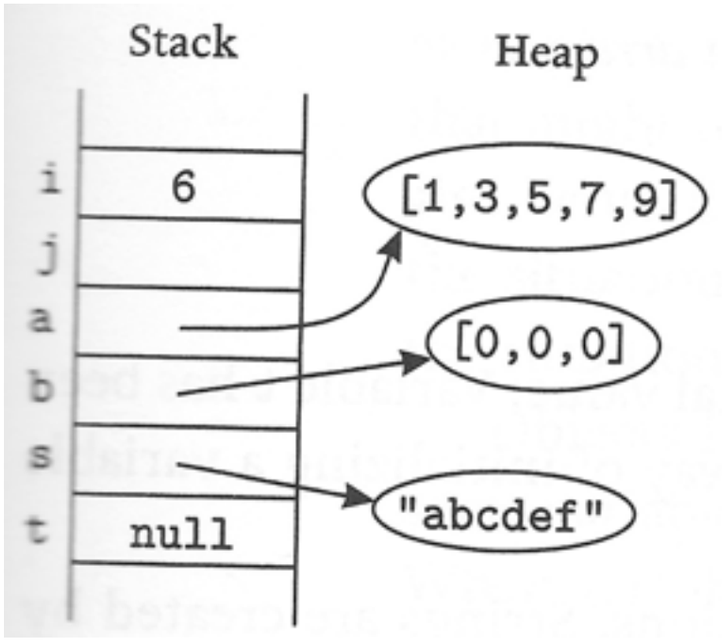
# Immutable Queue Algebraic Specs with Dynamic Methods

```
public static method:
empty  :              -> Queue

public dynamic methods:
enqueue: int -> Queue
dequeue:     -> Queue
first  :     -> int
size   :     -> int
isEmpty:     -> boolean

q.enqueue(i).dequeue()
   = q                                  if(q.isEmpty())
q.enqueue(i).dequeue()
   = q.dequeue().enqueue(i)             if(!q.isEmpty())

q.enqueue(i).first()
   = i               if(q.isEmpty())
q.enqueue(i).first()
   = q.first()       if(!q.isEmpty())

q.size(q.empty()) = 0
q.size(q.enqueue(i)) = 1 + q.size()

Queue.empty().isEmpty() = true
q.enqueue(i).isEmpty() = false

Queue.empty().toString() = ""
q.enqueue(i).toString() = q.toString() +"(" + i + ")"
```

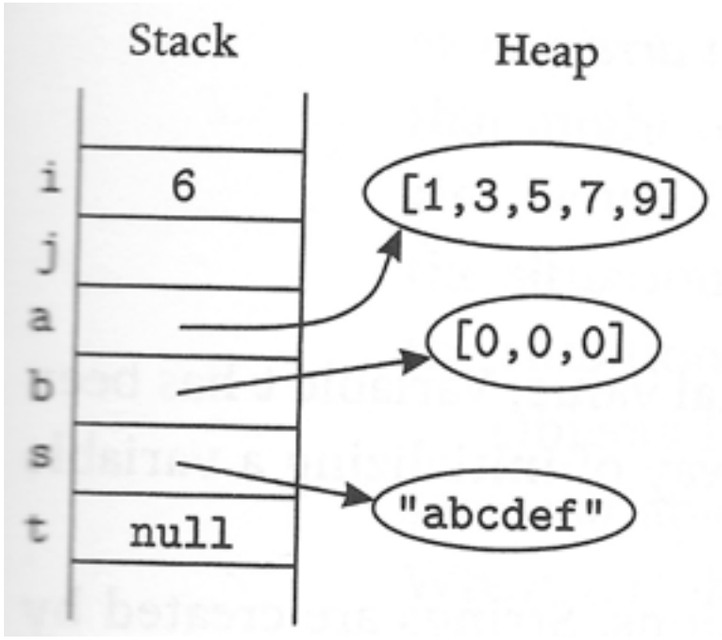
# Mutable Queue Specs

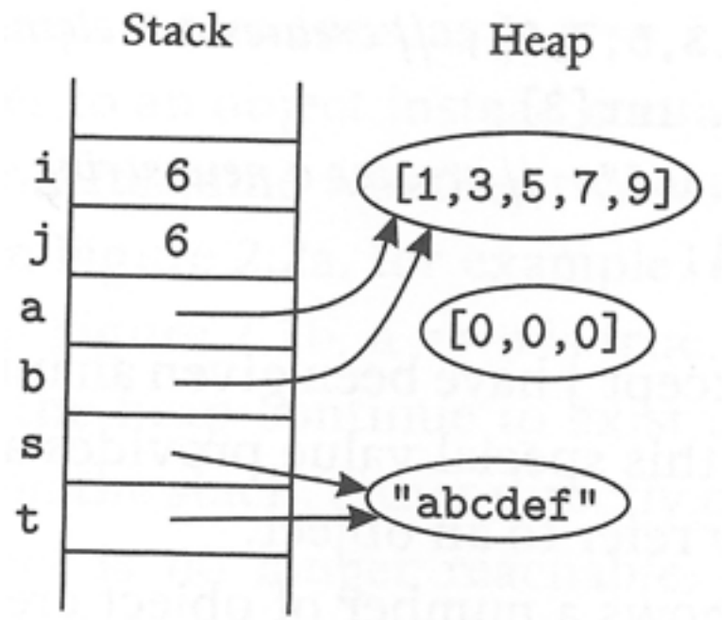
```
public static method:
empty  :                  -> Queue

public dynamic methods:
enqueue: int ->                adds int to end of
                               queue
dequeue:       ->              removes first int from
                               queue
first  :       -> int          returns first int in
                               queue
size   :       -> int          returns size of queue
                               (number of ints)
isEmpty:       -> boolean      returns whether Queue
                               is empty
```

```
int i = 6;

int j; //uninitialized

int [] a = {1, 3, 5, 7, 9}; //
creates a 5-element array

int [] b = new int[3];

String s = "abcdef";

String t = null;
```

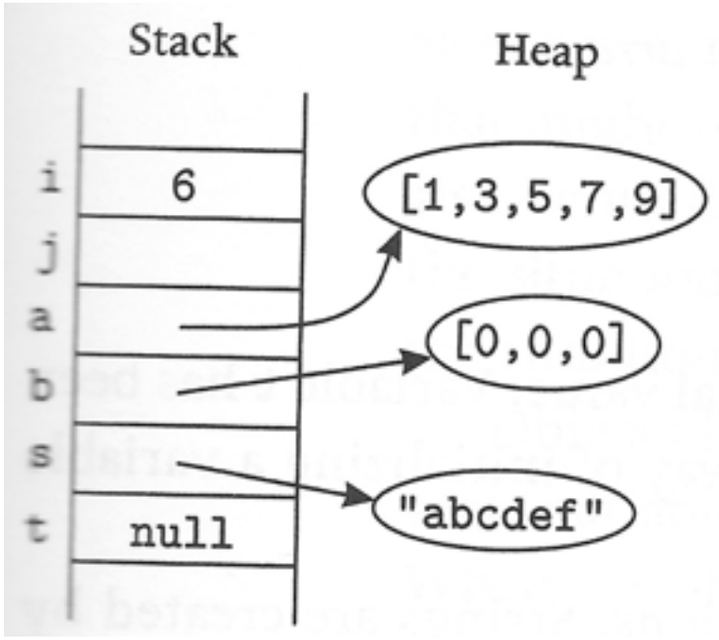


```
int i = 6;

int j; //uninitialized

int [] a = {1, 3, 5, 7, 9}; //
creates a 5-element array

int [] b = new int[3];

String s = "abcdef";

String t = null;


j = i;

b = a;

t = s;
```

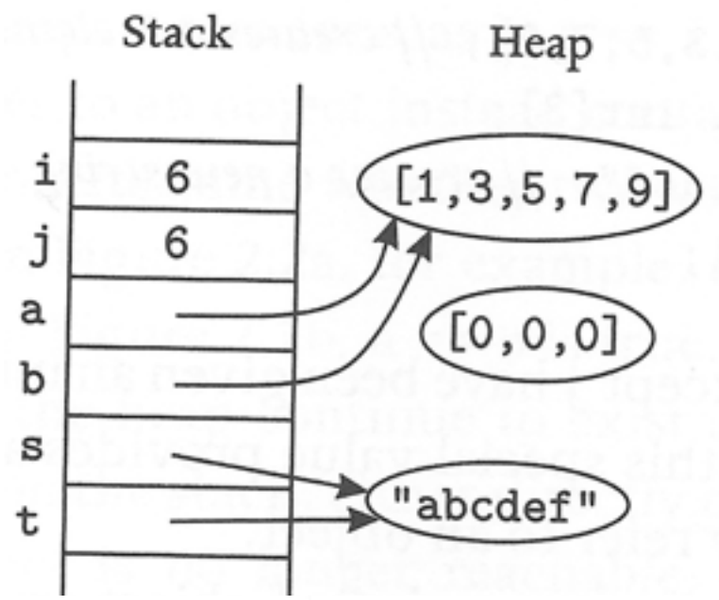


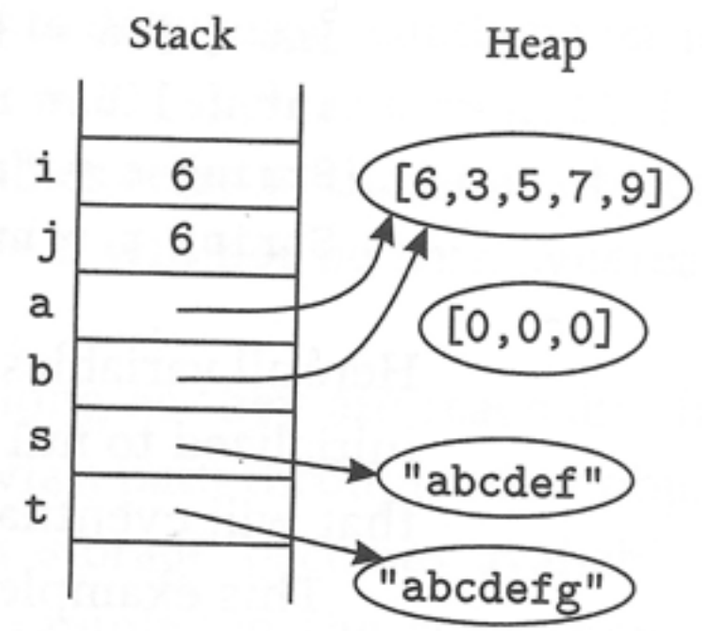
```
int i = 6;

int j; //uninitialized

int [] a = {1, 3, 5, 7, 9}; //
creates a 5-element array

int [] b = new int[3];

String s = "abcdef";

String t = null;
```




```
j = i;

b = a;

t = s;
```




```
t = t +"g";

a[i] = e;
```



# Understanding an implementation

- abstraction function
- representation invariant

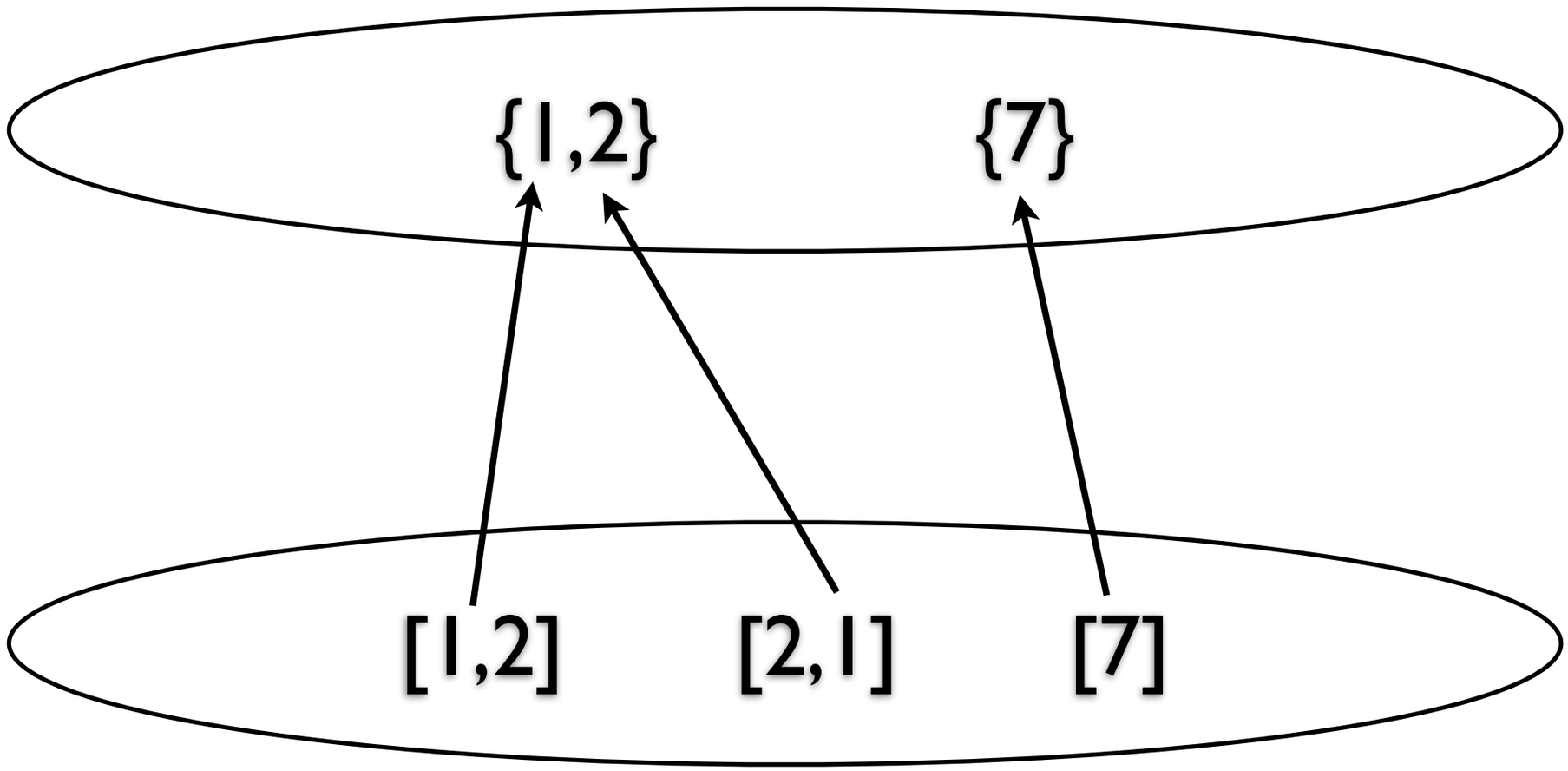
# Abstraction Function [Liskov]

This relationship can be defined by a function called the abstraction function that maps from the instance variables that make up the rep of an object to the abstract object being represented:

$$AF : C \rightarrow A$$

Specifically, the abstraction function AF males from a concrete state (i.e., the state of an object of the class C) to an abstract state (i.e., the state of an abstract object). For each object c belonging to C, AF(c) is the state of the abstract object $a \in A$ that c represents.

# IntSet Abstraction Function [Liskov]




AF(c) = {c.els[i].intValue | 0<=i<c.els.size}

# FSetString Recipe Implementation Abstraction Function

$$AF(FSetString.emptySet()) = \{\}$$

$$AF(FSetString.insert(s0, k0)) = \left\{ \begin{array}{ll} AF(s0) & |FSetString.contains(s0, k0) \\ k0 + AF(s0) & |else \end{array} \right\}.$$

# FSetString Recipe Implementation Abstraction Function

$$AF(EmptySet()) = \{\}$$

$$AF(Insert(s0, k0)) = \left\{ \begin{array}{ll} AF(s0) & |s0.contains(k0) \\ k0 + AF(s0) & |else \end{array} \right\}.$$

# Immutable Queue Abstraction Function

# Immutable Queue Abstraction Function

$$AF(Queue.empty()) = \text{“”}$$

$$AF(Queue.enqueue(q, i)) = q.toString() + \text{“(”} + i + \text{“)”}$$

# Immutable Queue with Dynamic Methods Abstraction Function

# Immutable Queue with Dynamic Methods Abstraction Function

$$AF(Queue.empty()) = \text{“”}$$

$$AF(q.enqueue(i)) = q.toString() + \text{“(”} + i + \text{“)”}$$

# Mutable Queue Abstraction Function

# Mutable Queue Implemented as ArrayList Abstraction Function

$$AF(Queue.empty()) = \text{“”}$$

$$AF(c) = \{\text{“(”} + c.q.get(i).intValue() + \text{“)”} | 0 <= i < c.q.size()\}$$

# Rep Invariant

A statement of a property that all legitimate objects satisfy is called a *representation invariant*, or *rep invariant*. A rep invariant I is a predicate

$$I : C \rightarrow \text{boolean}$$

that is true of legitimate objects.