Northeastern University

# Lecture 7: Bags - 1

Prof. Chen-Hsiang (Jones) Yu, Ph.D.

College of Engineering

Materials are edited by Prof. Jones Yu from

Data Structures and Abstractions with Java, 5th edition. By Frank M. Carrano and Timothy M. Henry.
ISBN-13 978-0-13-483169-5 

# Abstract Data Type (ADT)

# Computer Data Organization

- Abstract Data Type (ADT)
  - » A specification that describes a data set and the operations on that data
  - » Does not indicate how to store the data or how to implement the operations
  - » Independent of any programming language
- Data Structure
  - » An implementation of an ADT within a programming language

# Computer Data Organization (cont.)

- Collection
  - » Is a general term for an ADT
  - » Groups objects and provides various services to its client
  - » Enables a client to add, remove, retrieve, and query the objects it represents.
  - » Is an abstraction and is an abstract data type (ADT).
- Examples:
  - Bag
  - List
  - Stack
  - Queue
  - Dictionary
  - Tree
  - Graph

# Java Interlude 1 - Generics

# Generic Data Types

- Enable you to write a placeholder instead of an actual class type within the definition of a class or interface
- The placeholder is
  - A generic data type
  - A type parameter
- You define a generic class and the client chooses data type of the objects in collection.

# Interface

```
1 public interface Pairable<T>
2 {
3    public T getFirst();
4    public T getSecond();
5    public void changeOrder();
6 }  // end Pairable
```

Listing J1-1: The interface **`Pairable`**

# Example

```
1  /**
2     A class of ordered pairs of objects having the same data type.
3     @author Frank M. Carrano
4  */
5  public class OrderedPair<T> implements Pairable<T>
6  {
7     private T first, second;
8
9     public OrderedPair(T firstItem, T secondItem) // NOTE: no <T> after
10    {                                             // constructor name
11       first = firstItem;
12       second = secondItem;
13    } // end constructor
14
15    /** Returns the first object in this pair. */
16    public T getFirst()
17    {
18       return first;
19    } // end getFirst
20
21    /** Returns the second object in this pair. */
22    public T getSecond()
```

Listing J1-2: The class **`OrderedPair`**

# Example

```
20
21      /** Returns the second object in this pair. */
22      public T getSecond()
23      {
24         return second;
25      } // end getSecond
26
27      /** Returns a string representation of this pair. */
28      public String toString()
29      {
30         return "(" + first + ", " + second + ")";
31      } // end toString
32
33      /** Interchanges the objects in this pair. */
34      public void changeOrder()
35      {
36         T temp = first;
37         first = second;
38         second = temp;
39      } // changeOrder
40   } // end OrderedPair
```

Listing J1-2: The class **`OrderedPair`**

# Example: Creating `OrderedPair` object

```
OrderedPair<String> fruit
                = new OrderedPair<String>("apple", "banana");
```

**or**

```
OrderedPair<String> fruit
                  = new OrderedPair<>("apple", "banana");
```

# Exercise: Creating `OrderedPair` object

```
OrderedPair<String> fruit = new OrderedPair<>("apple", "banana");
System.out.println(fruit);
fruit.changeOrder();
System.out.println(fruit);
String firstFruit = fruit.getFirst()
System.out.println(firstFruit + " has length " + firstFruit.length());
```

output results

# Answer

```
OrderedPair<String> fruit = new OrderedPair<>("apple", "banana");
System.out.println(fruit);
fruit.changeOrder();
System.out.println(fruit);
String firstFruit = fruit.getFirst()
System.out.println(firstFruit + " has length " + firstFruit.length());
```

**(apple, banana)**

**(banana, apple)**

**banana has length 6**

# Bags

# The ADT Bag

- Definition
  - » A finite collection of objects in no particular order
  - » Can contain duplicate items
- Possible behaviors
  - » Get number of items
  - » Check for empty
  - » Add items
  - » Remove items

# CRC Card

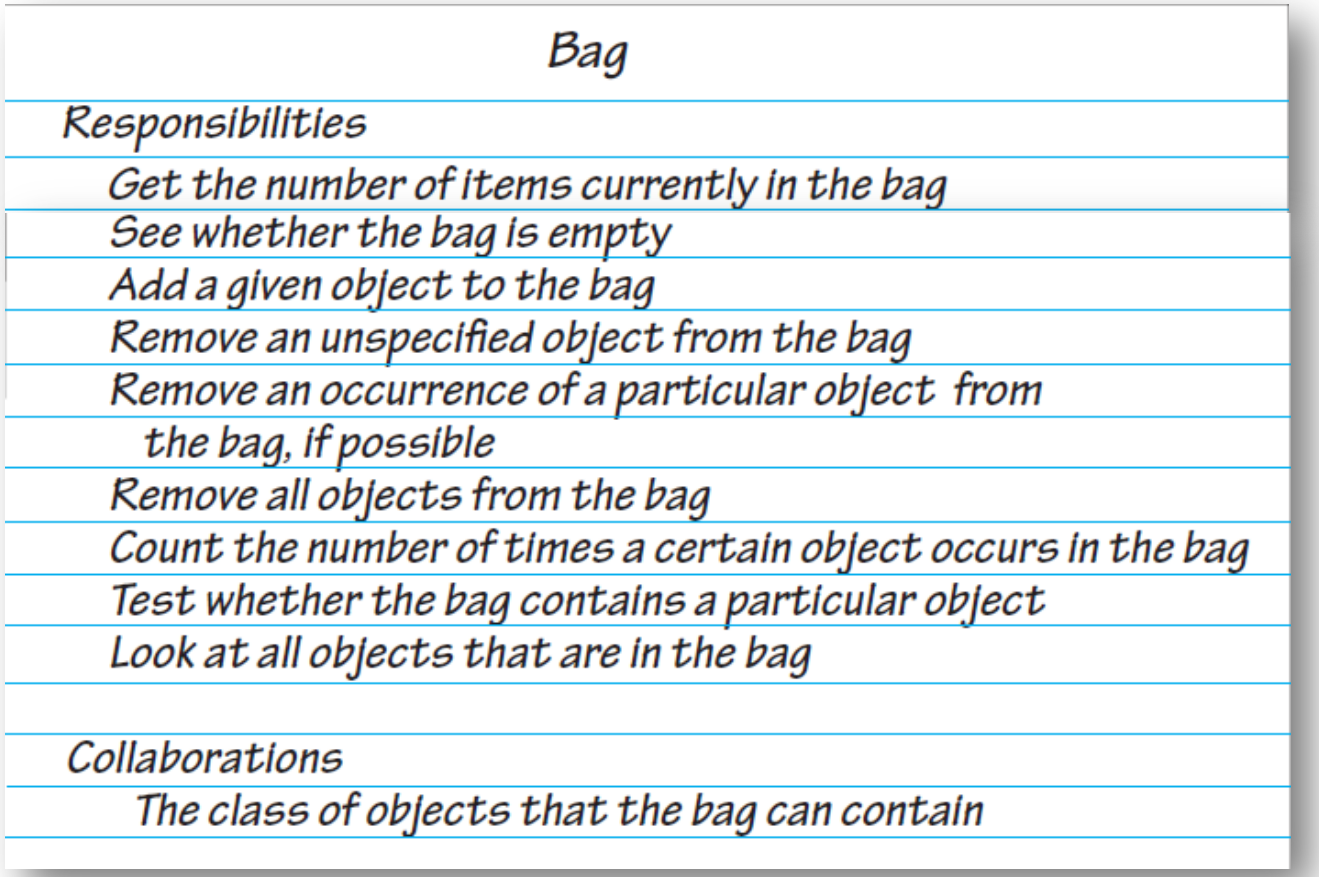


Figure 1-1: A CRC card for a class **Bag**

CRC: class-responsibility-collaboration

# Specifying a Bag

- Before implementing a bag in Java, we need to describe its data and specify the methods in details.
- Options we can take when `add` cannot complete its task:
  - » Do nothing
  - » Leave bag unchanged, but signal the client

# UML Notation

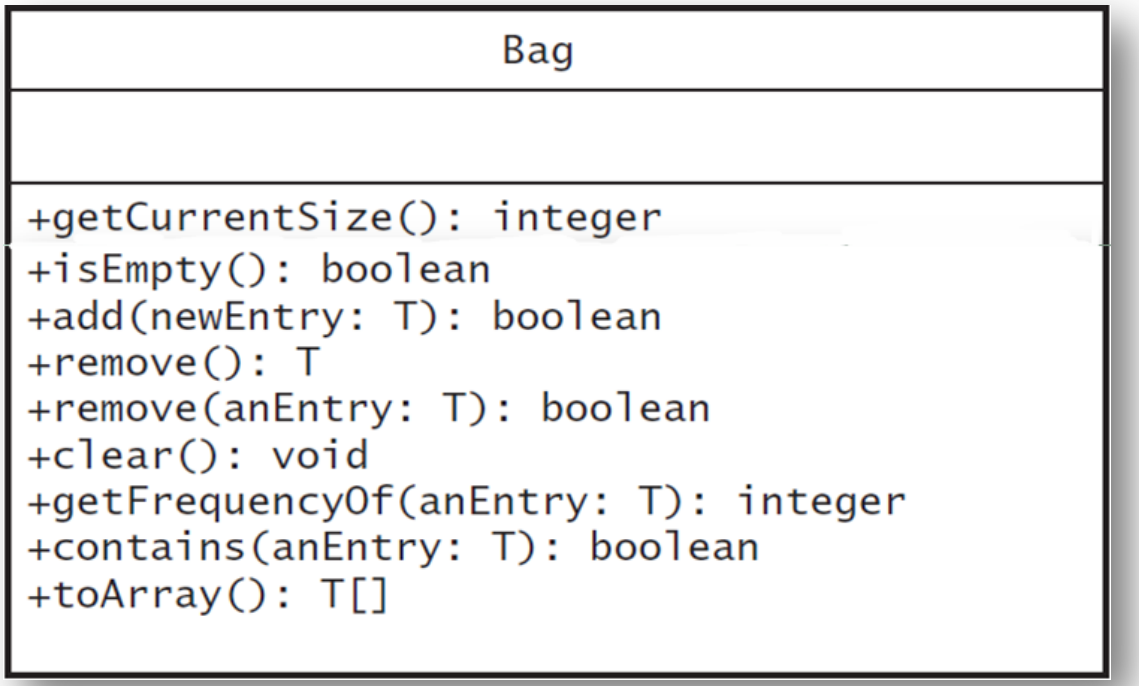


Figure 1-2: UML notation for the class **`Bag`**

# Design Decision

- What to do for unusual conditions?
  - » Assume it won't happen
  - » Ignore invalid situations
  - » Guess at the client's intention
  - » Return value that signals a problem
  - » Return a boolean
  - » Throw an exception

# Exercise

- Suppose aBag represents an empty bag that has a finite capacity.
- Write some pseudocode statements to add user-supplied strings to the bag until it becomes full.

# Answers

```
// aBag is empty

entry = next string read from the user

while ( aBag.add(entry) )
{

    entry = next string read from the user

}

// aBag is full
```

# Exercise

- Given the full bag aBag that you created in previous exercise, write some pseudocode statements that remove and display all the strings in the bag.

# Answers

```
// aBag is full

while ( !aBag.isEmpty() )
{
    entry = aBag.remove();
    display entry
}

// aBag is empty
```