Northeastern University

Lecture 8: Bags - 2

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: Bag

Abstract Data Type: Bag		
Data		
▪ A finite number of objects, not necessarily distinct, in no particular order, and having the same data type ▪ The number of objects in this collection		
Operations		
Pseudocode	UML	Description
`getCurrentSize()`	`+getCurrentSize(): integer`	Task: Reports the current number of objects in the bag. Input: None. Output: The number of objects currently in the bag.
`isEmpty()`	`+isEmpty(): boolean`	Task: Sees whether the bag is empty. Input: None. Output: True or false according to

		whether the bag is empty.
add(newEntry)	*+add(newEntry: T): boolean*	Task: Adds a given object to the bag. Input: *newEntry* is an object. Output: True or false according to whether the addition succeeds.
remove()	*+remove(): T*	Task: Removes an unspecified object from the bag, if possible. Input: None. Output: Either the removed object, if the removal was successful, or *null*.
remove(anEntry)	*+remove(anEntry: T): boolean*	Task: Removes one particular object from the bag, if possible. Input: *anEntry* is an object. Output: True or false according to whether the removal succeeds.
clear()	*+clear(): void*	Task: Removes all objects from the bag. Input: None. Output: None.
getFrequencyOf(anEntry)	*+getFrequencyOf(anEntry: T): integer*	Task: Counts the number of times an object occurs in the bag. Input: *anEntry* is an object. Output: The number of times *anEntry* occurs in the bag.
contains(anEntry)	*+contains(anEntry: T): boolean*	Task: Tests whether the bag contains a particular object. Input: *anEntry* is an object. Output: True or false according to whether *anEntry* occurs in the bag.
toArray()	*+toArray(): T[]*	Task: Retrieves all objects in the bag. Input: None. Output: A new array of entries currently in the bag.

An Interface

```java
/**
   An interface that describes the operations of a bag of objects.
   @author Frank M. Carrano
*/
public interface BagInterface<T>
{
   /** Gets the current number of entries in this bag.
       @return  The integer number of entries currently in the bag. */
   public int getCurrentSize();

```

Listing 1-1: A Java interface for a class of bags

An Interface

```
11    /** Sees whether this bag is empty.
12        @return  True if the bag is empty, or false if not. */
13    public boolean isEmpty();
14
15    /** Adds a new entry to this bag.
16        @param newEntry  The object to be added as a new entry.
17        @return  True if the addition is successful, or false if not. */
18    public boolean add(T newEntry);
19
20    /** Removes one unspecified entry from this bag, if possible.
21        @return  Either the removed entry, if the removal
22                 was successful, or null. */
23    public T remove();
24
25    /** Removes one occurrence of a given entry from this bag, if possible.
26        @param anEntry  The entry to be removed.
27        @return  True if the removal was successful, or false if not. */
28    public boolean remove (T anEntry);
29
30    /** Removes all entries from this bag. */
```

Listing 1-1: A Java interface for a class of bags

An Interface

```
25      /** Removes one occurrence of a given entry from this bag, if possible.
26          @param anEntry  The entry to be removed.
27          @return  True if the removal was successful, or false if not. */
28      public boolean remove (T anEntry);
29
30      /** Removes all entries from this bag. */
31      public void clear();
32
33      /** Counts the number of times a given entry appears in this bag.
34          @param anEntry  The entry to be counted.
35          @return  The number of times anEntry appears in the bag. */
36      public int getFrequencyOf(T anEntry);
37
38      /** Tests whether this bag contains a given entry.
39          @param anEntry  The entry to locate.
40          @return  True if the bag contains anEntry, or false if not. */
41      public boolean contains(T anEntry);
42
43      /** Retrieves all entries that are in this bag.
44          @return  A newly allocated array of all the entries in the bag.
45                   Note: If the bag is empty, the returned array is empty. */
46      public T[] toArray();
47  } // end BagInterface
```

Listing 1-1: A Java interface for a class of bags

Using the ADT Bag

```
1  /**
2      A class that maintains a shopping cart for an online store.
3      @author Frank M. Carrano
4  */
5  public class OnlineShopper
6  {
7     public static void main(String[] args)
8     {
9        Item[] items = {new Item("Bird feeder", 2050),
10                       new Item("Squirrel guard", 1547),
11                       new Item("Bird bath", 4499),
12                       new Item("Sunflower seeds", 1295)};
13       BagInterface<Item> shoppingCart = new Bag<>();
14       int totalCost = 0;
15
16       // Statements that add selected items to the shopping cart:
17       for (int index = 0; index < items.length; index++)
18       {
19          Item nextItem = items[index]; // Simulate getting item from shopper
20          shoppingCart.add(nextItem);
21          totalCost = totalCost + nextItem.getPrice();
22       } // end for
23
24       // Simulate checkout
25       while (!shoppingCart.isEmpty())
```

Listing 1-2: A program that maintains a bag for online shopping

Using the ADT Bag

```
24          // Simulate checkout
25          while (!shoppingCart.isEmpty())
26             System.out.println(shoppingCart.remove());
27
28          System.out.println("Total cost: " + "\t$" + totalCost / 100 + "." +
29                             totalCost % 100);
30       } // end main
31    } // end OnlineShopper
```

Output

```
Sunflower seeds $12.95
Bird bath       $44.99
Squirrel guard  $15.47
Bird feeder     $20.50
Total cost:     $93.91
```

Listing 1-2: A program that maintains a bag for online shopping

Example: A Piggy Bank

```
1  /**
2     A class that implements a piggy bank by using a bag.
3     @author Frank M. Carrano
4  */
5  public class PiggyBank
6  {
7    private BagInterface<Coin> coins;
8
9    public PiggyBank()
10   {
11     coins = new Bag<>();
12   } // end default constructor
13
14   public boolean add(Coin aCoin)
15   {
16     return coins.add(aCoin);
17   } // end add
```

Listing 1-3: A class of piggy banks

Example: A Piggy Bank

```
14    public boolean add(Coin aCoin)
15    {
16       return coins.add(aCoin);
17    } // end add
18
19    public Coin remove()
20    {
21       return coins.remove();
22    } // end remove
23
24    public boolean isEmpty()
25    {
26       return coins.isEmpty();
27    } // end isEmpty
28 } // end PiggyBank
```

Listing 1-3: A class of piggy banks

Example: A Piggy Bank

```
1  /**
2     A class that demonstrates the class PiggyBank.
3     @author Frank M. Carrano
4  */
5  public class PiggyBankExample
6  {
7     public static void main(String[] args)
8     {
9        PiggyBank myBank = new PiggyBank();
10
11       addCoin(new Coin(1, 2010), myBank);
12       addCoin(new Coin(5, 2011), myBank);
13       addCoin(new Coin(10, 2000), myBank);
14       addCoin(new Coin(25, 2012), myBank);
15
16       System.out.println("Removing all the coins:");
17       int amountRemoved = 0;
18
19       while (!myBank.isEmpty())
20       {
21          Coin removedCoin = myBank.remove();
22          System.out.println("Removed a " + removedCoin.getCoinName() + ".");
```

Listing 1-4: A demonstration of the class **PiggyBank**

Example: A Piggy Bank

```
19          while (!myBank.isEmpty())
20          {
21             Coin removedCoin = myBank.remove();
22             System.out.println("Removed a " + removedCoin.getCoinName() + ".");
23             amountRemoved = amountRemoved + removedCoin.getValue();
24          } // end while
25          System.out.println("All done. Removed " + amountRemoved + " cents.");
26       } // end main
27
28       private static void addCoin(Coin aCoin, PiggyBank aBank)
29       {
30          if (aBank.add(aCoin))
31             System.out.println("Added a " + aCoin.getCoinName() + ".");
32          else
33             System.out.println("Tried to add a " + aCoin.getCoinName() +
34                                ", but couldn't");
35       } // end addCoin
36    } // end PiggyBankExample
```

Listing 1-4: A demonstration of the class **PiggyBank**

Example: A Piggy Bank

Output

```
Added a PENNY.
Added a NICKEL.
Added a DIME.
Added a QUARTER.
Removing all the coins:
Removed a QUARTER.
Removed a DIME.
Removed a NICKEL.
Removed a PENNY.
All done. Removed 41 cents.
```

Listing 1-4: A demonstration of the class **`PiggyBank`**

Exercise

- A language-independent specification for a group of values and operations on those values is called a/an:
  - abstract data type
  - data structure
  - collection
  - primitive

Answer

- A language-independent specification for a group of values and operations on those values is called a/an:
  - abstract data type **
  - data structure
  - collection
  - primitive

Exercise

- Which behavior is not represented in a bag?
  - reorder the bag
  - report the number of items in the bag
  - report if the bag is empty
  - add an item to the bag

Answer

- Which behavior is not represented in a bag?
  - reorder the bag **
  - report the number of items in the bag
  - report if the bag is empty
  - add an item to the bag

Using ADT Like Using Vending Machine

Figure 1-3: A vending machine

Observations about Vending Machines

- Can perform only tasks machine's interface presents.
- You must understand these tasks
- Cannot access the inside of the machine
- You can use the machine even though you do not know what happens inside.
- Usable even with new insides.

Observations about ADT Bag

- Can perform only tasks specific to ADT
- Must adhere to the specifications of the operations of ADT
- Cannot access data inside ADT without ADT operations
- Use the ADT, even if don't know how data is stored
- Usable even with new implementation

Sets

The ADT Set

- A special kind of bag, one that does not allow repeated, or duplicate, entries.
- Most of the bag's operations are the same for the ADT set, but need to modify `add` and `remove`.
- Furthermore, no need to have `getFrequencyOf`. (Why?)
- We can use `contains` method instead.

```java
/** An interface that describes the operations of a set of objects. */
public interface SetInterface<T>
{
   public int getCurrentSize();
   public boolean isEmpty();

   /** Adds a new entry to this set, avoiding duplicates.
       @param newEntry  The object to be added as a new entry.
       @return  True if the addition is successful, or
                false if the item already is in the set. */
   public boolean add(T newEntry);

   /** Removes a specific entry from this set, if possible.
       @param anEntry  The entry to be removed.
       @return  True if the removal was successful, or false if not. */
   public boolean remove(T anEntry);

   public T remove();
   public void clear();
   public boolean contains(T anEntry);
   public T[] toArray();
} // end SetInterface
```

Listing 1-5: A Java interface for a class of sets

Java Class Library: The Interface `Set`

- Belong to package java.util within Java Class Library
- Sets that adhere to the specification do not contain a pair of equal objects x and y, i.e., `x.equals(y)` is true.
- There are differences between interfaces `Set` and `SetInterface`.

Exercise

- Please download "L8_E1" from the "In-class Exercise" folder in Canvas.
- The project has "Bag.java" and "BagInterface.java" files where `Bag` class implements `BagInteface` interface.
- Create a testing program (MyBag.java) that creates a Bag object to add and remove some fruits:
  - add "Apple", add "Banana", add "Orange", remove "Banana", add "Apple"
- Print out: (1) the total number of fruits in the bag, and (2) the total number of "Apple" in the bag

Answer

```
public class MyBag {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        BagInterface<String> mybag = new Bag<>();

        mybag.add("Apple");
        mybag.add("Banana");
        mybag.add("Orange");
        mybag.remove("Banana");
        mybag.add("Apple");

        System.out.println("Total number in bag is: "
                           + mybag.getCurrentSize());

        System.out.println("Total number of Apple is: "
                           + mybag.getFrequencyOf("Apple"));
    }
}
```

```
Total number in bag is: 3
Total number of Apple is: 2
```