Bags

WIA1002/ WIB1002 :
Data structure

What items?

What operations?

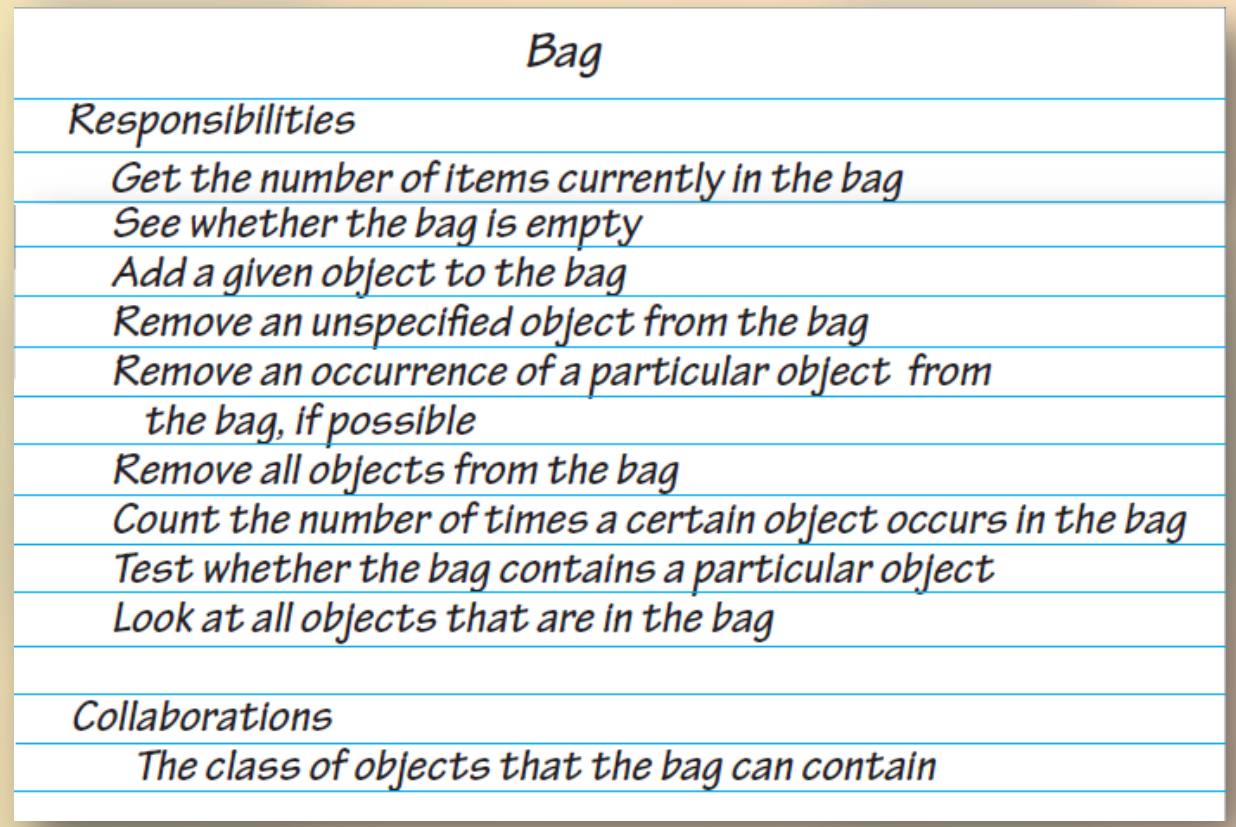
Yes or No?

- Should the items be stored in a specific order?
- Can you keep repetitive items in the same bag?
- Is there a standard limit/number of items to be stored in the bag?

The ADT Bag

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

CRC Card



Bag
Responsibilities
Get the number of items currently in the bag
See whether the bag is empty
Add a given object to the bag
Remove an unspecified object from the bag
Remove an occurrence of a particular object from the bag, if possible
Remove all objects from the bag
Count the number of times a certain object occurs in the bag
Test whether the bag contains a particular object
Look at all objects that are in the bag
Collaborations
The class of objects that the bag can contain

FIGURE 1-1 A CRC card for a class **Bag**

Specifying a Bag

- Describe its data and specify in detail the methods that correspond to the bag's behaviors.
- Name the methods, choose their parameters, decide their return types, and write comments to fully describe their effect on the bag's data.

UML Notation

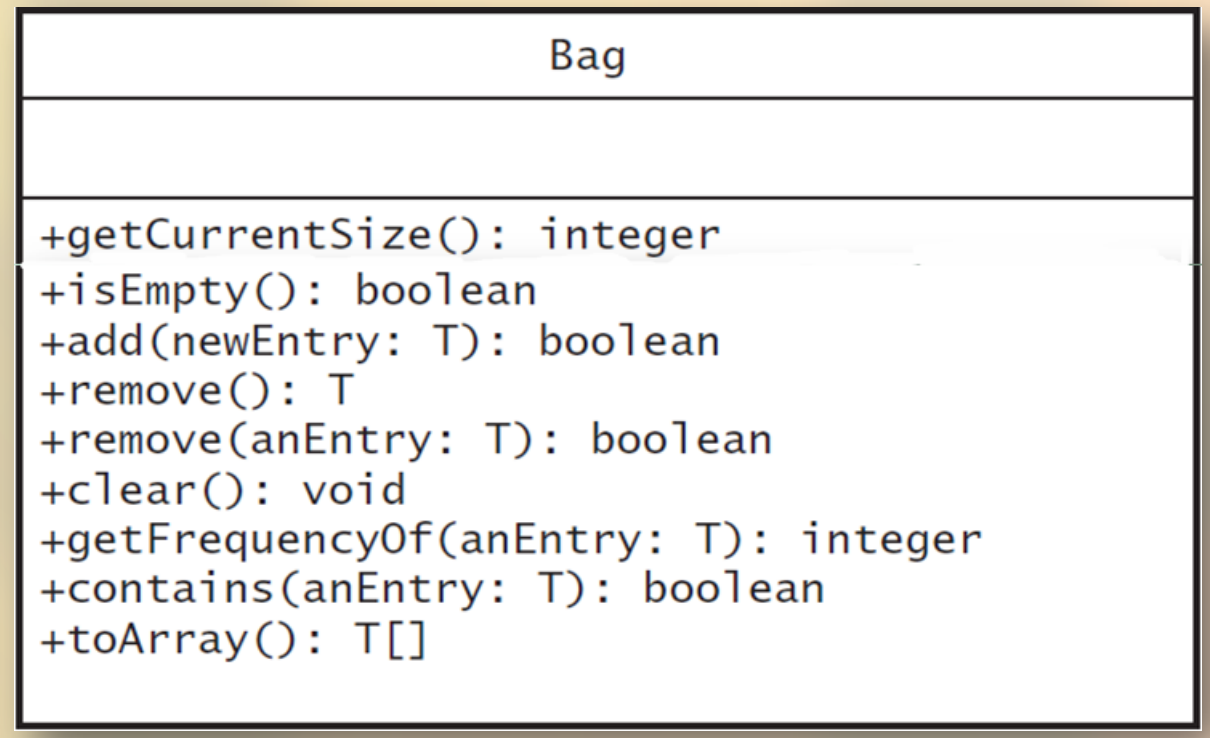


FIGURE 1-2 UML notation for the class **Bag**

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

An Interface

- can write Java headers for the bag's methods and organize them into a Java *interface* for the class that will implement the ADT.

```
1  /**
2     An interface that describes the operations of a bag of objects.
3     @author Frank M. Carrano
4  */
5  public interface BagInterface<T>
6  {
7     /** Gets the current number of entries in this bag.
8         @return  The integer number of entries currently in the bag. */
9     public int getCurrentSize();
10
```

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

Implementing the ADT bag

- Imagine we hire a programmer to implement the ADT bag in Java, given the interface and specifications that we have developed.
- We do not need to know *how* the programmer implemented the bag to be able to use it. We only need to know *what* the ADT bag does.

Using the ADT bag

- So, assume that we have a Java class, *Bag*, that implements the Java interface *BagInterface*
- Two examples on how we can use Bag: OnlineShopper and PiggyBank

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)
```

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

Using the ADT Bag

```
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`**

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.

Java Class Library: The Interface `Set`

```
1  /** An interface that describes the operations of a set of objects. */
2  public interface SetInterface<T>
3  {
4     public int getCurrentSize();
5     public boolean isEmpty();
6
7     /** Adds a new entry to this set, avoiding duplicates.
8         @param newEntry  The object to be added as a new entry.
9         @return  True if the addition is successful, or
10                 false if the item already is in the set. */
11    public boolean add(T newEntry);
12
13    /** Removes a specific entry from this set, if possible.
14        @param anEntry  The entry to be removed.
15        @return  True if the removal was successful, or false if not. */
16    public boolean remove(T anEntry);
```

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

Java Class Library: The Interface `Set`

```
13      /** Removes a specific entry from this set, if possible.
14          @param anEntry  The entry to be removed.
15          @return  True if the removal was successful, or false if not. */
16      public boolean remove(T anEntry);
17
18      public T remove();
19      public void clear();
20      public boolean contains(T anEntry);
21      public T[] toArray();
22  } // end SetInterface
```

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

Additional Slides

- Bag implemented using array (ArrayBag)
- Bag implemented using list (LinkedBag)

Fixed-Size Array (ArrayBag)

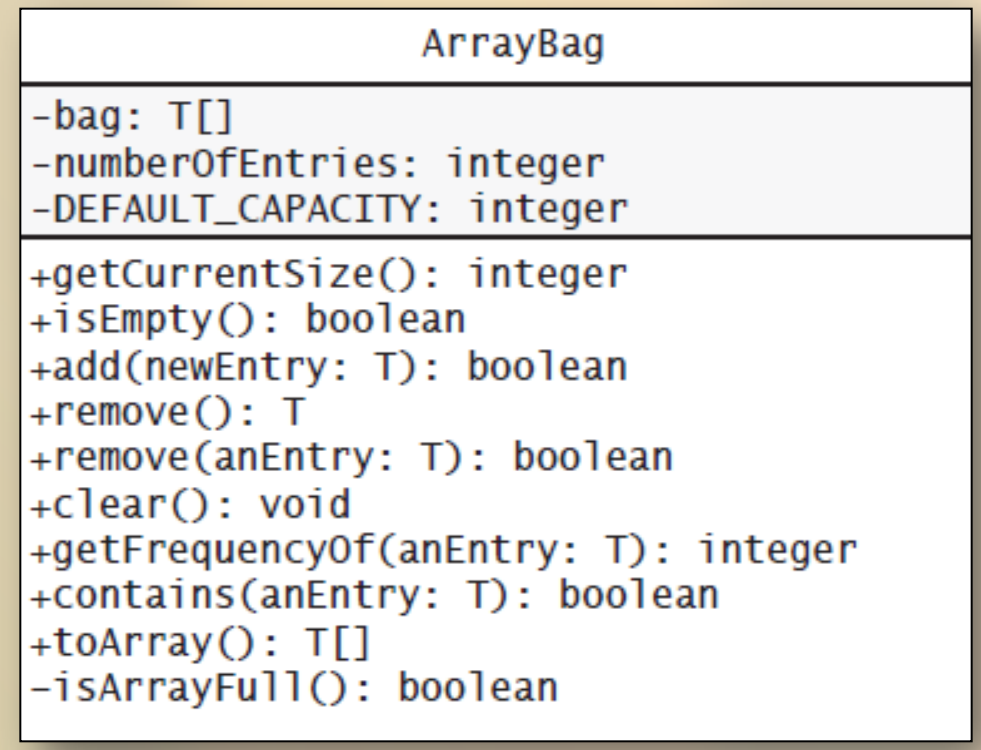


FIGURE 2-2 UML notation for the class **`ArrayBag`**, including the class's data fields

Fixed-Size Array (ArrayBag)

```
1  /**
2     A class of bags whose entries are stored in a fixed-size array.
3     @author Frank M. Carrano
4  */
5  public final class ArrayBag<T> implements BagInterface<T>
6  {
7     private final T[] bag;
8     private int numberOfEntries;
9     private static final int DEFAULT_CAPACITY = 25;
10
11    /** Creates an empty bag whose initial capacity is 25. */
12    public ArrayBag()
13    {
14        this(DEFAULT_CAPACITY);
15    } // end default constructor
16
17    /** Creates an empty bag having a given initial capacity.
18        @param capacity  The integer capacity desired. */
19    public ArrayBag(int capacity)
```

LISTING 2-1 An outline of the class **`ArrayBag`**

Note : When a class header includes an implements clause, the class must define all of the methods in the interface.

Fixed-Size Array (ArrayBag)

```
17    /** Creates an empty bag having a given initial capacity.
18        @param capacity  The integer capacity desired. */
19    public ArrayBag(int capacity)
20    {
21        // The cast is safe because the new array contains null entries.
22        @SuppressWarnings("unchecked")
23        T[] tempBag = (T[])new Object[capacity]; // Unchecked cast
24        bag = tempBag;
25        numberOfEntries = 0;
26    } // end constructor
27
28    /** Adds a new entry to this bag.
29        @param newEntry  The object to be added as a new entry.
30        @return  True if the addition is successful, or false if not. */
31    public boolean add(T newEntry)
32    {
33        < Body to be defined >
34    } // end add
35
36    /** Retrieves all entries that are in this bag.
```

LISTING 2-1 An outline of the class **ArrayBag**

Fixed-Size Array

```
36    /** Retrieves all entries that are in this bag.
37        @return  A newly allocated array of all the entries in the bag. */
38    public T[] toArray()
39    {
40        < Body to be defined >
41    } // end toArray
42
43    // Returns true if the arraybag is full, or false if not.
44    private boolean isArrayFull()
45    {
46        < Body to be defined >
47    } // end isArrayFull
48
49    < Similar partial definitions are here for the remaining methods
50      declared in BagInterface. >
51
52    . . .
53 } // end ArrayBag
```

LISTING 2-1 An outline of the class **ArrayBag**

An Outline of the Class `LinkedBag`

```
1  /**
2      A class of bags whose entries are stored in a chain of linked nodes.
3      The bag is never full.
4      @author Frank M. Carrano
5  */
6  public final class LinkedBag<T> implements BagInterface<T>
7  {
8     private Node firstNode;       // Reference to first node
9     private int  numberOfEntries;
10
11    public LinkedBag()
12    {
13       firstNode = null;
14       numberOfEntries = 0;
15    } // end default constructor
16
17    // Implementations of the public methods declared in BagInterface go here.
```

LISTING 3-2 An outline of the class `LinkedBag`

An Outline of the Class `LinkedBag`

```
14          numberOfEntries = 0;
15      } // end default constructor
16
17      < Implementations of the public methods declared in BagInterface go here. >
18
19      . . .
20
21      private class Node // Private inner class
22      {
23          < See Listing 3-1. >
24      } // end Node
25  } // end LinkedBag
```

LISTING 3-2 An outline of the class `LinkedBag`

Reference

- Chapter 1, *Data Structures and Abstractions with Java, 3e/4e,* Frank Carrano
- Additional slides, Chapter 2, *Data Structures and Abstractions with Java, 3e/4e,* Frank Carrano