# Bags – Part I

- References:
  - Text book : Chapter 1 and Chapter 2
  - Oracle/Sun Java Tutorial : http://java.sun.com/j2se/1.5.0/docs/guide/collections/index.html

## 1. ADT Bags

- Let consider a bag containing finite number of items in no particular order, example : shopping bags, backpacks, handbags, etc. A bag can contain duplicate item and it is a kind of ***container***.

- Possible operations: bag is empty or full, add new item, remove an item, remove all items, count how many items, display all items and etc

- Specifying ADT Bag: describe data, specify methods for bag's behaviors - Name methods, choose parameters, decide return types and write comments

  Data:
  - A finite number of object, not necessary distinct, in no particular order, and having the same data type
  - The number of objects

  Methods:

| Bag |
| --- |
| |
| +getCurrentSize(): integer<br>+isFull(): boolean<br>+isEmpty(): boolean<br>+add(newEntry: T): boolean<br>+remove(): T<br>+remove(anEntry: T): boolean<br>+clear(): void<br>+getFrequencyOf(anEntry: T): integer<br>+contains(anEntry: T): boolean<br>+toArray(): T[] |

Notes :

- ADT does not specify "how" to implement
- ADT specifies what methods/operations are available to use

    - To decide primitive operations, a designer should determine "what a user can do to a collection of data".
    - If more primitive operations are needed later, you can always update Bag ADT to include more operations.

- Design Decisions

    - What should the method add do when it cannot add a new entry?
        - Nothing?
        - Leave bag unchanged, signal client of condition?

    - What should happen when an unusual condition occurs?
        - Assume invalid never happens?
        - Ignore invalid event?
        - Return flag value?
        - Return boolean value – success/failure?
        - Throw exception?

- After refining operations, we may use Java interface to specify ADT Bag methods. Note:

    Interface has no data fields, constructors
    Methods must be public
    T is generic type;
    Specify and explain purpose, parameters, return type

```
/**
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 ();

    /** Sees whether this bag is full.
    @return true if the bag is full, or false if not */
    public boolean isFull ();

    /** Sees whether this bag is empty.
    @return true if the bag is empty, or false if not */
    public boolean isEmpty ();

    /** Adds a new entry to this bag.
    @param newEntry the object to be added as a new entry
    @return true if the addition is successful, or false if not */
    public boolean add (T newEntry);

    /** Removes one unspecified entry from this bag, if possible.
    @return either the removed entry, if the removal
    was successful, or null */
    public T remove ();

    /** Removes one occurrence of a given entry from this bag,
    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);

    /** Removes all entries from this bag. */
    public void clear ();

    /** Counts the number of times a given entry appears in this bag.
    @param anEntry the entry to be counted
    @return the number of times anEntry appears in the bag */
    public int getFrequencyOf (T anEntry);

    /** Tests whether this bag contains a given entry.
    @param anEntry the entry to locate
    @return true if the bag contains anEntry, or false otherwise */
    public boolean contains (T anEntry);

    /** Creates an array of all entries that are in this bag.
    @return a newly allocated array of all the entries in the bag */
    public T [] toArray ();

} // end BagInterface
```

- A bag object should be viewed as a container (collection of data) with a set of methods (as defined in interface).

  - Clients can only perform operations in ADT
  - Clients cannot access container directly without using ADT operations
  - Clients do not need to know implementation details
  - If implementation is changed, as long as interface is the same, client still use bag in same way as before

- Using ADT Bag: Assume that we have implemented BagInterface (to be done in next couple of chapters), we can now use our Bag class.

## Example 1: Shopping cart

```
/**
A class that maintains a shopping cart for an online store.
@author Frank M. Carrano
*/
public class OnlineShopper
{
   public static void main (String [] args)
   {    // simulate items to be added into shopping cart
      Item [] items = {new Item ("Bird feeder", 2050),
         new Item ("Squirrel guard", 1547),
         new Item ("Bird bath", 4499),
         new Item ("Sunflower seeds", 1295) };

      BagInterface < Item > shoppingCart = new Bag < Item > ();
      int totalCost = 0;

      // statements that add selected items to the shopping cart:
      for (int index = 0 ; index < items.length ; index++)
      {
         Item nextItem = items [index]; // simulate getting item from shopper
         shoppingCart.add (nextItem);
         totalCost = totalCost + nextItem.getPrice ();
      } // end for

      // simulate checkout
      while (!shoppingCart.isEmpty ())
         System.out.println (shoppingCart.remove ());
      System.out.println ("Total cost: " +
            "\t$" + totalCost / 100 + "." +
            totalCost % 100);
   } // end main

} // end OnlineShopper
```

Output:

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

Example2: Use a bag object to implement new class, PiggyBank (this is called adapter class)

```
/**
A class that implements a piggy bank by using a bag.
@author Frank M. Carrano
*/
public class PiggyBank
{
   private BagInterface < Coin > coins;
   public PiggyBank ()
   {
      coins = new Bag < Coin > ();  // A bag object
   } // end default constructor

   public boolean add (Coin aCoin)
   {
      return coins.add (aCoin);
   } // end add

   public Coin remove ()
   {
      return coins.remove ();
   } // end remove

   public boolean isEmpty ()
   {
      return coins.isEmpty ();
   } // end isEmpty
} // end PiggyBank

// Assume a class Coin is available with info:  amount, name, year etc

/**
A class that demonstrates the class PiggyBank.
@author Frank M. Carrano
*/
public class PiggyBankExample
{
   public static void main (String [] args)
   {
      PiggyBank myBank = new PiggyBank ();
```

```
        addCoin (new Coin (1, 2010), myBank);
        addCoin (new Coin (5, 2011), myBank);
        addCoin (new Coin (10, 2000), myBank);
        addCoin (new Coin (25, 2012), myBank);

        System.out.println ("Removing all the coins:");
        int amountRemoved = 0;

        while (!myBank.isEmpty ())
        {
           Coin removedCoin = myBank.remove ();
           System.out.println ("Removed a " + removedCoin.getCoinName () +
                 ".");
           amountRemoved = amountRemoved + removedCoin.getValue ();
        } // end while
        System.out.println ("All done. Removed " + amountRemoved +
              " cents.");
     } // end main

     private static void addCoin (Coin aCoin, PiggyBank aBank)
     {
        if (aBank.add (aCoin))
           System.out.println ("Added a " + aCoin.getCoinName () + ".");
        else
           System.out.println ("Tried to add a " + aCoin.getCoinName () +
                 ", but couldn't");
     } // end addCoin

  } // end PiggyBankExample
```

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
```

- Java Class Library: The Interface **Set**

  - The standard package contains an interface, Set, similar to bag interface
  - It does not allow duplicate items
  - Several methods provided:

| Method Summary | |
|---|---|
| boolean | **add**(E e)<br>Adds the specified element to this set if it is not already present (optional operation). |
| void | **clear**()<br>Removes all of the elements from this set (optional operation). |
| boolean | **contains**(Object o)<br>Returns true if this set contains the specified element. |
| boolean | **equals**(Object o)<br>Compares the specified object with this set for equality. |
| boolean | **isEmpty**()<br>Returns true if this set contains no elements. |
| boolean | **remove**(Object o)<br>Removes the specified element from this set if it is present (optional operation). |
| int | **size**()<br>Returns the number of elements in this set (its cardinality). |
| Object[] | **toArray**()<br>Returns an array containing all of the elements in this set. |

Complete listing :
http://download.oracle.com/javase/6/docs/api/java/util/Set.html

## Example to use Java API Set:

```
import java.util.*;

public class SetExample {

   public static void main(String[] args) {

      //Set example using TreeSet class
      Set<String> s=new TreeSet<String>();

      // add data
      s.add("bbbb"); s.add("aaaa"); s.add("ddddd"); s.add("cc");

      System.out.println("Number of Set data:"+s.size());

      // use for each loop
      for (String item: s)
         System.out.print(item+" ");
      System.out.println();

      System.out.println();

      // remove data
      s.remove("aaaa"); s.remove("cc");

      System.out.println("Number of Set data:"+s.size());
      for (String item: s)
         System.out.print(item+" ");
      System.out.println();

   }
}
```

```
$ java SetExample
Number of Set data:4
aaaa bbbb cc ddddd

Number of Set data:2
bbbb ddddd
```

## 2. Bag Implementation That Use Arrays

- We will see several ways to implements Bag ADT (or interface)
- In this section, we will use fixed-size array, then use resizing array
- Strategy: always fill slots from beginning and no "empty" slot in between 2 used slots

  Suppose I have the following letters from position 1 to 10, and some available spaces:

  A D G I B K O P C S _ _ _ _ _ _ _ _ _ _

  If I want to add a new letter E, then it is easy to add to first available space

  A D G I B K O P C S E _ _ _ _ _ _ _ _ _

  If I want to remove letter I from 4th position, it is easy to move last item to replace item I

  A D G E B K O P C S _ _ _ _ _ _ _ _ _

- Private data fields for implementation of **ArrayBag**

  Implements interface **BagInterface** of last section

  ```
  private final T [] bag; // array of bag entries
  private static final int DEFAULT_CAPACITY = 25;
  private int numberOfEntries; // current number of entries in bag
  ```

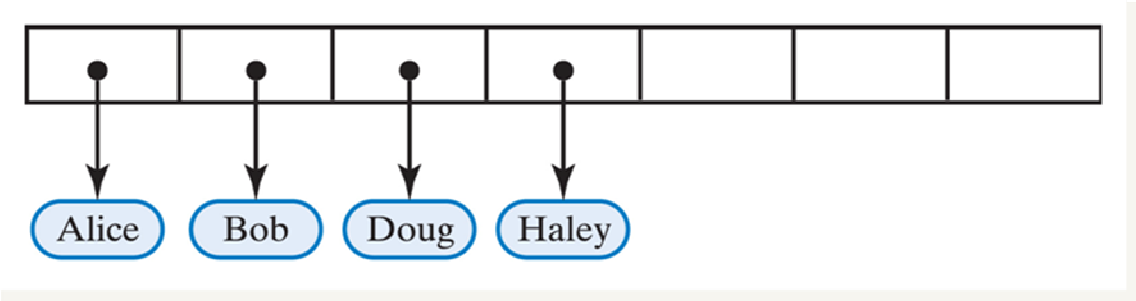
- Note: Java array index : 0,1,…,MAX_SIZE-1

- Outline of source code:

```
public class ArrayBag < T > implements BagInterface < T >
{
   private final T [] bag;
   private static final int DEFAULT_CAPACITY = 25;
   private int numberOfEntries;

   /** Creates an empty bag whose initial capacity is 25. */
   public ArrayBag ()
   {
      this (DEFAULT_CAPACITY);
   } // end default constructor


   /** Creates an empty bag having a given initial capacity.
   @param capacity the integer capacity desired */

   public ArrayBag (int capacity)
   {
      numberOfEntries = 0;
      // the cast is safe because the new array contains null entries
      @ SuppressWarnings ("unchecked")
      T [] tempBag = (T []) new Object [capacity];
      bag = tempBag;
   } // end constructor


   /** Adds a new entry to this bag.
   @param newEntry the object to be added as a new entry
   @return true if the addition is successful, or false if not */

   public boolean add (T newEntry)
   {
      //Body to be defined
   } // end add
```

```
/** Retrieves all entries that are in this bag.
@return a newly allocated array of all the entries in the bag */
public T [] toArray ()
{
   // Body to be defined
} // end toArray


/** Sees whether this bag is full.
@return true if the bag is full, or false if not */
public boolean isFull ()
{
   // Body to be defined
} // end isFull


// Similar partial definitions are here for the remaining methods
//    declared in BagInterface.
//  ...

} // end ArrayBag
```

- Example: Array of objects, each entry contains reference to an object. move object means change reference of an object




- Strategy:  identify group of core methods: Define, Test, Then finish rest of class

- isFull method:

```
/** Sees whether this bag is full.
    @return true if the bag is full, or false if not */
public boolean isFull()
{
   return numberOfEntries == bag.length;
} // end isFull
```

- toArray method:

```
/** Retrieves all entries that are in this bag.
    @return a newly allocated array of all the entries in the bag */
public T[] toArray()
{
   // the cast is safe because the new array contains null entries
   @SuppressWarnings("unchecked")
   T[] result = (T[])new Object[numberOfEntries]; // unchecked cast
   for (int index = 0; index < numberOfEntries; index++)
   {
      result[index] = bag[index];
   } // end for

   return result;
} // end toArray
```

  - Should **toArray** return the array **bag** or a copy?
  - Best to return a copy … think about why.

- Add method:
  - Assign new entry at end of array
  - Increment length of bag
  - Set returning Boolean value

```
/** Adds a new entry to this bag.
    @param newEntry  the object to be added as a new entry
    @return true if the addition is successful, or false if not */
public boolean add(T newEntry)
{
   boolean result = true;
   if (isFull())
   {
      result = false;
   }
   else
   {  // assertion: result is true here
      bag[numberOfEntries] = newEntry;
      numberOfEntries++;
   } // end if

   return result;
} // end add
```

- Temporarily make stub methods (for incomplete methods) for testing at this stage. Example: public boolean isEmpty() { return false;}

```
public class ArrayBagDemo1
{
   public static void main (String [] args)
   {
      // a bag that is not full
      BagInterface < String > aBag = new ArrayBag < String > ();

      aBag.add("aaa"); aBag.add("cccc");
      displayBag(aBag);
      if (aBag.isFull())
               System.out.println ("a full bag:");
      else
               System.out.println ("a bag that is not full:");

      aBag = new ArrayBag < String > (3);
      aBag.add("aaa"); aBag.add("cccc");aBag.add(ddd);
      displayBag(aBag);
      if (aBag.isFull())
               System.out.println ("a full bag:");
      else
               System.out.println ("a bag that is not full:");

   } // end main

   // Tests the method toArray while displaying the bag.
   private static void displayBag (BagInterface < String > aBag)
   {
      System.out.println ("The bag contains the following string(s):");
      Object [] bagArray = aBag.toArray ();
      for (int index = 0 ; index < bagArray.length ; index++)
      {
         System.out.print (bagArray [index] + " ");
      } // end for
      System.out.println ();
   } // end displayBag
} // end ArrayBagDemo1
```

- More Methods

```
/** Sees whether this bag is empty.
    @return true if the bag is empty, or false if not */
public boolean isEmpty()
{
   return numberOfEntries == 0;
} // end isEmpty

/** Gets the current number of entries in this bag.
    @return the integer number of entries currently in the bag */
public int getCurrentSize()
{
   return numberOfEntries;
} // end getCurrentSize

/** Counts the number of times a given entry appears in this bag.
    @param anEntry  the entry to be counted
    @return the number of times anEntry appears in the bag */
public int getFrequencyOf(T anEntry)
{
   int counter = 0;

   for (int index = 0; index < numberOfEntries; index++)
   {
      if (anEntry.equals(bag[index]))
      {
         counter++;
      } // end if
   } // end for

   return counter;
} // end getFrequencyOf

/** Tests whether this bag contains a given entry.
    @param anEntry  the entry to locate
    @return true if the bag contains anEntry, or false otherwise */
public boolean contains(T anEntry)
{
   boolean found = false;

   for (int index = 0; !found && (index < numberOfEntries); index++)
   {
      if (anEntry.equals(bag[index]))
      {
         found = true;
      } // end if
   } // end for

   return found;
} // end contains
```

- Method that remove items

// May also consider call remove() each entry to clear

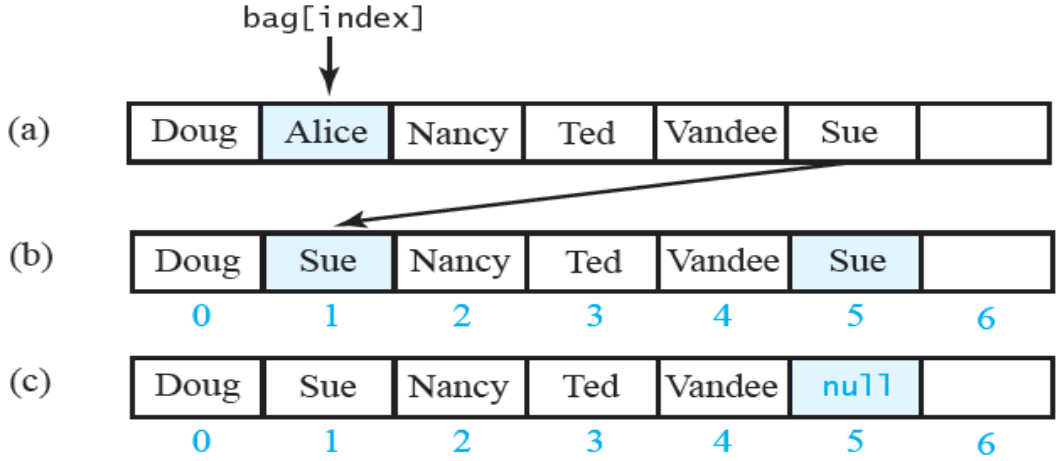
```
public void clear()
{
   numberOfEntries = 0;
} // end clear
```

// remove last entry

```
public T remove()
{
   T result = null;
   if (numberOfEntries > 0)
   {
      result = bag[numberOfEntries - 1];
      bag[numberOfEntries - 1] = null;
      numberOfEntries--;
   } // end if

   return result;
} // end remove
```

// To remove specific item

1. Need to search for the desired item
2. Move last item to replace the desired item position (more efficient way)



Method must also handle error situations

- When desired item is not found
- When the bag is empty

```
/** Removes one occurrence of a given entry from this bag.
    @param anEntry  the entry to be removed
    @return true if the removal was successful, or false if not */
public boolean remove(T anEntry)
{
   int index = getIndexOf(anEntry);
   T result = removeEntry(index);
   return anEntry.equals(result);
} // end remove
```

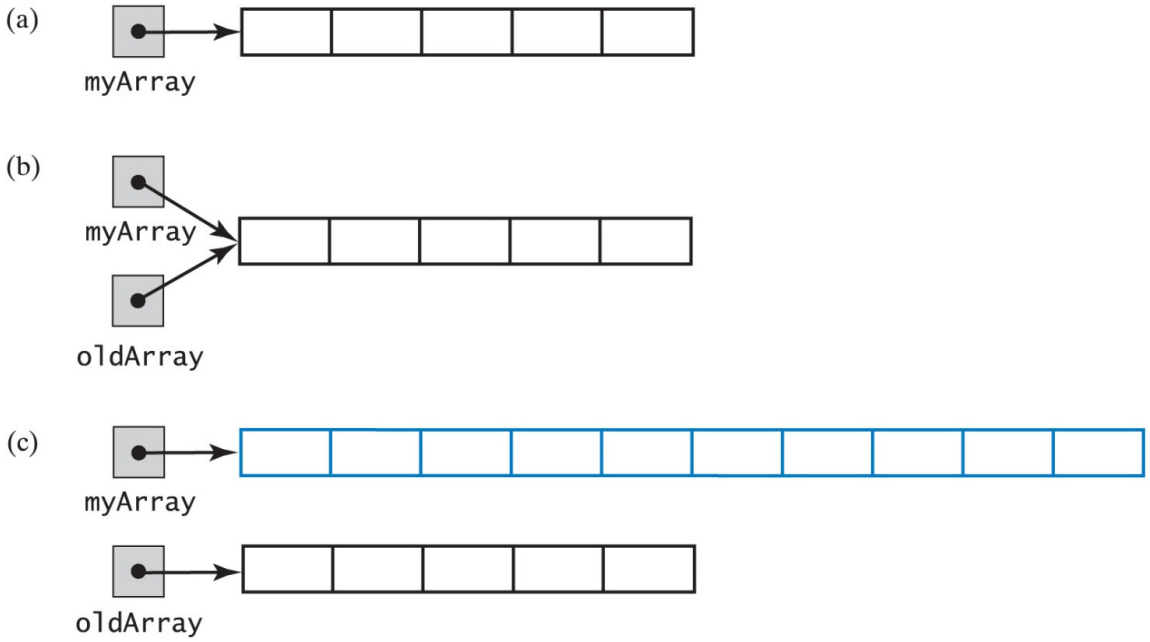
```
// getIndexOf() similar to contains(), but return index of the desired entry
// return -1 to indicate "Not found

private T removeEntry(int givenIndex)
{
     T result = null; // result to return

     if (!isEmpty()&& (givenIndex>= 0))
     {
       result = bag[givenIndex];  // remember entry here
       numberOfEntries--;
       bag[givenIndex]=bag[numberOfEntried]
     } // end if

     return result;
} // end getEntry
```

- How to expand an Array to handle flexible size

  - An array has a fixed size. If we need a larger bag, we are in trouble
  - Popular solution: When array becomes full

    Move its contents to a larger array (dynamic expansion)
    Copy data from original to new location
    Manipulate names so new location keeps name of original array

    Example:
    (a) int[] myArray = new int[initial_size]
    (b) int[] oldArray = myArray
    (c) myArray = new int[new_size] // bigger array
    (d) copy data from oldArray to myArray :
    for (int i=0; i<=oldArray.length;i++) myArray[i]=oldArray[i];




- There is also a predefined static method copyOf() in class Arrays:

// Copies the specified array, truncating or padding with zeros (if necessary) so the copy has the specified length.

static T[] **copyOf**(T[] original, int newLength)

- Use array expansion to implement Bag. Just update add() methods in previous implementation.

  Notes:
  - add() never return false
  - isFull() never return false
  - private T [] bag;  // remove "final" modifier

  Code segment:

```
public boolean add (T newEntry)
{
   if (numberOfEntries == bag.length)
      bag = Arrays.copyOf(bag, 2*bag.length);
   // add new entry after last current entry
   bag [numberOfEntries] = newEntry;
   numberOfEntries++;
   return true;
} // end add
```