Collection class

Read textbook Chapter 3 (3.1, 3.2)

Outline

- From object to collection
- Java Array
- Collection class: bag

Collection Classes

- A Collection class is a data type that is capable of holding a group of items.
- In Java, Collection classes can be implemented as a class, along with methods to add, remove, and examine items.

Java Array

```
int[] numbers;
int[] numbers = new int[4];
int[] numbers = new int[]{1,2,3};
```

Array operation

 How many components in the int array numbers?

Array operation

 How many components in the int array numbers?

numbers.length

Array operation

 How many components in the int array numbers?
 numbers.length

```
Q1: int[] numbers;
```

Q2: int[] numbers = new int[4];

Array operation (cont.)

- Assignment
- As parameter
- Exceptions
 - NullPointerException
 - ArrayIndexOutOfBoundsException
- Iterating over an array
 - for-loop
 - Use control variable

Outline

- From object to collection
- Java Array
- Collection class: bag

Bag Operations

- Each bag can be put in its initial state, which is an empty bag.
- You can check how many numbers are in the bag.
- You may check how many occurrences of a certain number are in the bag.
- Numbers can be added into the bag.
- Numbers can be removed from the bag.

public class IntArrayBag

```
{
...
}
```

The heading of the definition.

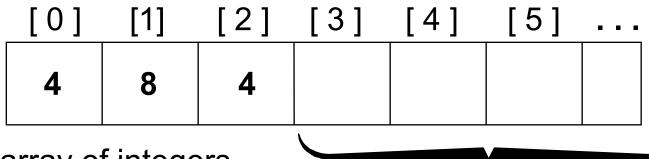
public class IntArrayBag

1. Instance variables

}

Implementation: instance variables

- The entries of a bag will be stored in the front part of an array, as shown in this example.
- The entries may appear in any order.



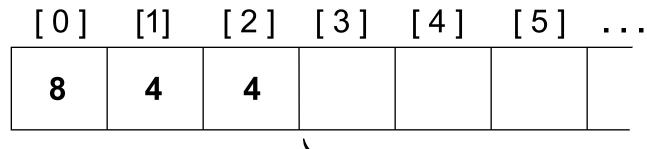
An array of integers

We don't care what's in this part of the array.

Implementation: instance variables

 We also need to keep track of how many numbers are in the bag.

An integer to keep track of the bag's size



An array of integers

We don't care what's in cs272, Huiping this part of the array.

An Exercise

 Use these ideas to write a list of private instance variables could implement the Bag class.

```
public class IntArrayBag
{
    private int[] data;
    private int manyItems;
    ...
}
```

public class IntArrayBag

```
...
```

1. Instance variables

```
public class IntArrayBag

{
    ...
    ...
    ...
    2. Constructors
```

```
public class IntArrayBag
 public IntArrayBag(){...}
 ...
```

1. Instance variables

2. Constructors

The Bag's Constructor

```
public IntArrayBag( )
public IntArrayBag(int initialCapacity)
```

Copy constructor public IntArrayBag(Object obj)

Specification: the Bag's Constructor

```
/* @postcondition
* This bag is empty and has an initial capacity of 10.
* @exception OutOfMemoryError */
public IntArrayBag( )
   final int INITIAL_CAPACITY = 10;
   manyItems = 0;
   data = new int[INITIAL_CAPACITY];
Final variable: the value will never be changed while the program
is running
```

Specification: the Bag's Constructor

```
/** @precondition
    initialCapacity is non-negative.
 * @postcondition
    This bag is empty and has the given initial capacity.
 * @exception IllegalArgumentException
    Indicates that initial Capacity is negative.
 * @exception OutOfMemoryError
*/
public IntArrayBag(int initialCapacity)
```

Implementation: the Bag's Constructor

```
public IntArrayBag(int initialCapacity)
{
    if (initialCapacity < 0)
        throw new IllegalArgumentException
        ("The initialCapacity is negative: " + initialCapacity);
        data = new int[initialCapacity];
        manyItems = 0;
}</pre>
```

Copy Constructor

```
public IntArrayBag(Object obj)
    - abnormal case process
        obj is not null
        obj is an instance of IntArrayBag
    - copy what?
        All the instance variables (num, array)
        Special attention needs to be paid to array copying
         Code:
         IntArrayBag toCopy = (IntArrayBag)obj;
        num = toCopy.num;
        data = new int[num];
        for(int i=0; i<num;i++)
             data[i] = toCopy.data[i];
```

```
public class IntArrayBag implements Cloneable
                                              1. Instance variables
 public IntArrayBag( ){...}
                                                 2. Constructors
 public void add(int element)
                                                   3. Methods
```

Accessor methods (Size related)

 Counts how many integers are in the bag. public int size()

Get the current capacity of this bag.
 public int getCapacity()

 Counts how many copies of a number occur public int countOccurrences(int target)

Size()

```
public int size( )
   return manyltems;
public int getCapacity( )
   return data.length:
```

countOccurrences

```
/* @Postcondition: The return value is the number of copies of
target in the Bag. */
public int countOccurrences(int target){
   int answer = 0;
   int index;
   for (index = 0; index < manyItems; index++){
      if (target == data[index])
          answer++;
   return answer;
```

Modification methods (add and remove)

- public void add(int element)
- public boolean remove(int target)
- Others
- public void addMany(int... elements)
- public void addAll(IntArrayBag newbag)

Specification: the add method

```
IntArrayBag bag1 = new IntArrayBag();
bag1.add(1);
/**
* @postcondition
  A new copy of the element has been added to this bag.
*/
public void add(int element){...}{
   //Make sure there is room for a new entry in the array.
    //Place newEntry in the appropriate location of the data array.
    //Add one to the instance variable manyItems.
```

Specification: the add method

```
/* @postcondition
```

- * A new copy of the element has been added to this bag.
- * @exception OutOfMemoryError
- * Indicates insufficient memory for increasing the bag's capacity.
- * @note An attempt to increase the capacity beyond
- * Integer.MAX_VALUE will cause the bag to fail with an
- * arithmetic overflow.

*/

public void add(int element)

Add method

```
public void add(int element){
   if (manyItems == data.length){
    //Ensure twice as much space as we need. And copy all the old conten
    int[] biggerArray = new int[(manyItems + 1)*2];
    System.arraycopy(data, 0, biggerArray, 0, manyItems);
    data = biggerArray;
   data[manyItems] = element;
   manyltems++;
System.arraycopy(src, sindex, dest, dindx, n);
```

ensureCapacity

```
public void <a href="mailto:ensureCapacity">ensureCapacity</a>(int minimumCapacity)
   int[] biggerArray;
   if (data.length < minimumCapacity)
     biggerArray = new int[minimumCapacity];
     System.arraycopy(data, 0, biggerArray, 0, manyItems);
     data = biggerArray;
```

Add method

```
public void add(int element)
   if (manyItems == data.length)
   { //Ensure twice as much space as we need.
    ensureCapacity((manyItems + 1)*2);
   data[manyItems] = element;
   manyltems++;
```

Using the Bag in a Program

 Here is typical code from a program that uses the new Bag class:

```
Bag ages = new Bag();

//Record the ages of three children:
ages.add(4);
ages.add(8);
ages.add(4);
```

Specification: the Remove Method

Removes one copy of a number

```
/* @ Postcondition: If target was in the Bag, then
* one copy of target has been removed from the
* Bag, and the return value is true; otherwise the
* Bag is unchanged and the return value is false.
*/
public boolean remove(int target)
{
...
}
```

Remove

```
public boolean remove(int target){
   int index; //The location of target in the data array.
   for (index = 0; (index < manyltems) && (target != data[index]); index++)
   //The target was not found, so nothing is removed.
   if (index == manyItems)
    return false;
   else{ //The target was found at data[index].
     //Reduce manyItems by 1 and copy the last element onto data[index].
     manyltems--;
     data[index] = data[manyItems];
    return true;
```

The ADT invariant

- The number of elements in the bag is stored in the instance variable "manyItems". manyItems<=data.length
- For an empty bag, do not care what is stored in data.
- For a non-empty bag, the elements are stored in data[0],..., data[manyltems-1]

Static method - Union

```
/*@precondition
  * Neither b1 nor b2 is null, and
  * b1.getCapacity() + b2.getCapacity() <= Integer.MAX VALUE.
  * @return the union of b1 and b2
  * @exception NullPointerException.
  * Indicates that one of the arguments is null.
  * @exception OutOfMemoryError
  * Indicates insufficient memory for the new bag.
  * @note
  * An attempt to create a bag with a capacity beyond
  * Integer.MAX_VALUE will cause an arithmetic overflow
  * that will cause the bag to fail. Such large collections should use
  * a different bag implementation.
  */
public static IntArrayBag union(IntArrayBag b1, IntArrayBag b2)
```

Implementation of Union method

```
public static IntArrayBag union(IntArrayBag b1, IntArrayBag b2)
    IntArrayBag answer = new IntArrayBag(b1.getCapacity() +
                             b2.getCapacity());
   System.arraycopy(b1.data, 0, answer.data, 0, b1.manyItems);
   System.arraycopy(b2.data, 0, answer.data, b1.manyItems, b2.manyItems);
   answer.manyltems = b1.manyltems + b2.manyltems;
   return answer;
```

Clone method (S.S.)

```
public IntArrayBag clone( )
 { // Clone an IntArrayBag object.
   IntArrayBag answer = null;
   try{
     answer = (IntArrayBag) super.clone( );
   }catch (CloneNotSupportedException e){
     throw new RuntimeException
        ("This class does not implement Cloneable");
    answer.data = data.clone();
    return answer;
```

Other Types of Bags

- In this example, we have implemented a bag containing integers.
- But we could have a bag of float numbers, a bag of characters, a bag of Strings . . .
- Suppose you wanted one of these other bags. How much would you need to change in the implementation?

Sequence ADT

- Constructor
- Accessor methods
 - Get size?
 - Get the first element?
 - Get the current element?
 - Get the next element?
- Modification methods
 - addBefore
 - addAfter
 - removeCurrent
 - Union \rightarrow concatenation

Summary

- A Collection class is a class that can hold a group of items.
- Collection classes can be implemented with a Java class.
- New java knowledge
 - Final variable
 - System.arraycopy()

References

- Clone method explanation: pages 134-136 (S.S.)
- Clone method code: pages 139 (S.S.)
- ensureCapacity method: pages 136, 139

Time analysis

Operation	Time Analysis
Constructor	O(c) c is the initial capacity
add	O(1) or O(n)
b1.addAll(b2)	O(n2) or (n1+n2)
clone	O(c)
countOfOccurrences	O(n)
getCapacity	O(1)
remove	O(n)
size	O(1)
union of b1 and b2	O(c1+c2)