Bag ADT
Learning Objectives

- Define an ADT for a data structure
- Implement an ADT in multiple ways
- Use connected Nodes to help store data in a structured way
- Evaluate the benefits of different implementations of an interface
Data Structures

- An object type used to store, retrieve, and represent data
  
  - **String**:  
    - A data structure consisting of an ordered sequence of characters (chars)  
    - Includes many methods (substring, charAt, length, etc.)
  
  - **Arrays**:  
    - Ordered, fixed-length sequence of data of a single type  
    - Methods: none that we’ve used!
A Data Structure of Our Own

- Each type of data structure has its own uses and limitations
  - We use Strings and arrays in different contexts
- We can define a data structure based on what we want it to do
  - Selecting its behaviors → choosing its methods
  - Defining a type based only on its behaviors is exactly the use case for an Abstract Data Type
The Bag Data Structure

- An unordered data structure that can contain only objects of a single type.
- Models how a physical bag can store objects
  - What behaviors does a bag have from the perspective of a storage device?
Modeling with Bag

- **Bag** objects are used to **store several elements of the same type**
- A **Bag** has a **variable size**: it grows or shrinks automatically to fit its contents
- Unlike our ordered data structures, elements in a bag do not have a **position**!
  - A bag contains a value or it does not: **membership**, not **order**.
# Book Bag Behavior

<table>
<thead>
<tr>
<th>Behavior</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count elements</td>
<td>How many elements are held in this bag?</td>
</tr>
<tr>
<td>Remove a particular element</td>
<td>Look for a particular element in the bag and remove it.</td>
</tr>
<tr>
<td>Remove a random element</td>
<td>Reach into the bag and pull out an element at random.</td>
</tr>
<tr>
<td>Check membership</td>
<td>Is a given element found among the elements contained in the bag?</td>
</tr>
<tr>
<td>Check emptiness</td>
<td>Does the bag have any elements at all?</td>
</tr>
<tr>
<td>Describe</td>
<td>Generate a text/String description of the contents of this bag</td>
</tr>
<tr>
<td>Behavior</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Count elements</td>
<td>How many elements are held in this bag?</td>
</tr>
<tr>
<td>Remove a particular</td>
<td>Look for a particular element in the bag and remove it.</td>
</tr>
<tr>
<td>element</td>
<td></td>
</tr>
<tr>
<td>Remove a random element</td>
<td>Reach into the bag and pull out an element at random.</td>
</tr>
<tr>
<td>Check membership</td>
<td>Is a given element found among the elements contained in the bag?</td>
</tr>
<tr>
<td>Check emptiness</td>
<td>Does the bag have any elements at all?</td>
</tr>
<tr>
<td>Describe</td>
<td>Generate a text/String description of the contents of this bag</td>
</tr>
</tbody>
</table>
Headers in the Bag ADT

- The ADT provided has some new terms in the method headers:
  - You don’t have to follow this directly in the assignments, but you might find the keywords helpful when writing these headers
- **@param**: an input parameter
- **@return**: a description of what value to return
- **@precondition**: something that is assumed to be true of the object/input before calling the method
- **@postcondition**: something that is assumed to be true of the object/output after calling the method
Bag Methods: size()

- The size method returns the number of elements inside a Bag
- An empty Bag has a size of 0

Example:
Bag bookBag = new ArrayBag(); or
Bag bookBag = new NodeBag();

System.out.println(bookBag.size()); ➞ 0
Bag Methods: add(element)

- The `add(element)` method adds the parameter element to the bag.
- Adding an element to the bag will increase the size of the bag by 1.
Bag Methods: \texttt{add(element)}

Example:

```
Bag bookBag = new ArrayBag();
System.out.println(bookBag.size()); ➙ 0
```
**Bag Methods: add(element)**

Example:

```java
Bag bookBag = new ArrayBag();
System.out.println(bookBag.size()); ➞ 0
bookBag.add(gilead);
System.out.println(bookBag.size()); ➞ 1
```

The elements must be of type `Book`
Bag Methods: `add(element)`

Example:
```java
Bag bookBag = new ArrayBag();
System.out.println(bookBag.size()); // 0
bookBag.add(gilead);
System.out.println(bookBag.size()); // 1
```

The elements must be of type `Book`
Bag Methods: `add(element)`

Example:

```java
Book bookBag = new Bag();
bookBag.add(gilead);
```
Bag Methods: `add(element)`

Example:

```java
Book bookBag = new Bag ();
bookBag.add(gilead);

bookBag.add(actualAir);
```
Bag Methods: `add(element)`

Example:

```java
Book bookBag = new Bag ();
bookBag.add(gilead);

bookBag.add(actualAir);

bookBag.add(outline);
```
Bag Methods: \texttt{remove}(target)

- The \texttt{remove}(target) method removes one element from the bag that’s equal to the target.
- The method will need to move all the elements in the implementation after the removal to fill up the gap.
- The method decreases the size of the bag by 1.
List Methods: remove(index)

Example:
Book bookBag = new Bag();
bookBag.add(gilead);
bookBag.add(actualAir);
bookBag.add(outline);

studentsList.remove(actualAir);