CSE 21
Intro to Computing II

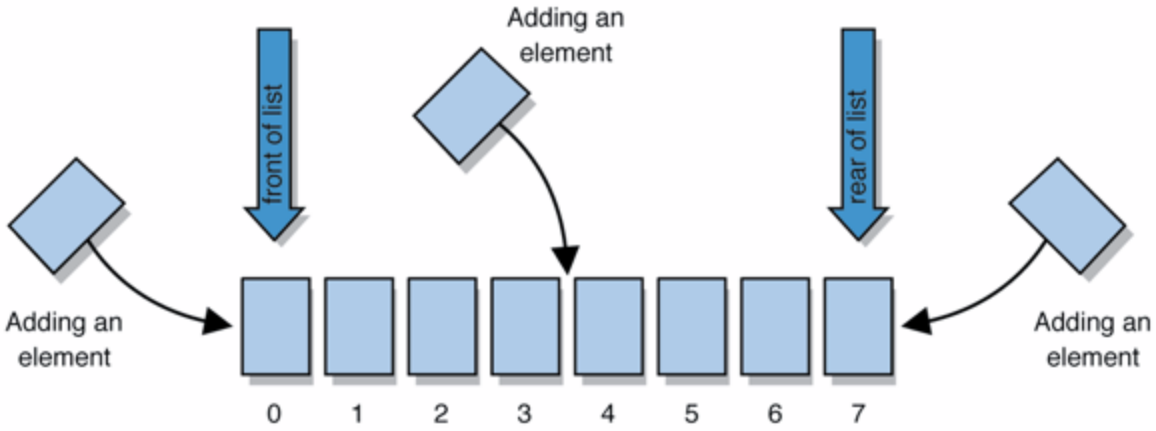
Lecture 10 – ArrayList

Announcement

- Lab#9 due before start of next lab
  - Type your answers in a text file and submit it as an attachment
- Reading assignment
  - Chapter 7.11 to 7.14, 12.1 to 12.6 of textbook

List of Objects

- An ordered sequence of elements:
  - each element is accessible by a 0-based **index**
  - a list has a **size** (number of elements that have been added)
  - elements can be added to the front, back, or elsewhere
  - in Java, a list can be represented as an **ArrayList** object



Contents of a List

- Rather than creating an array of boxes, create an object that represents a "list" of items. (initially an empty list.)
  - { }
- You can add items to the list.
  - The default behavior is to add to the end of the list.
    - {first, second, third, name}
- The list object keeps track of the element values that have been added to it, their order, indexes, and its total size.
  - Think of an "array list" as an automatically resizing array object.
  - Internally, the list is implemented using an array and a size field.

ArrayList Methods (1)

`add(`**`value`**`)`	appends value at end of list
`add(`**`index, value`**`)`	inserts given value just before the given index, shifting subsequent values to the right
`clear()`	removes all elements of the list
`indexOf(`**`value`**`)`	returns first index where given value is found in list (-1 if not found)
`get(`**`index`**`)`	returns the value at given index
`remove(`**`index`**`)`	removes/returns value at given index, shifting subsequent values to the left
`set(`**`index, value`**`)`	replaces value at given index with given value
`size()`	returns the number of elements in list
`toString()`	returns a string representation of the list such as "[3, 42, -7, 15]"

ArrayList Methods (2)

`addAll(`**`list`**`)` `addAll(`**`index, list`**`)`	adds all elements from the given list to this list (at the end of the list, or inserts them at the given index)
`contains(`**`value`**`)`	returns true if given value is found somewhere in this list
`containsAll(`**`list`**`)`	returns true if this list contains every element from given list
`equals(`**`list`**`)`	returns true if given other list contains the same elements
`iterator()` `listIterator()`	returns an object used to examine the contents of the list (seen later)
`lastIndexOf(`**`value`**`)`	returns last index value is found in list (-1 if not found)
`remove(`**`value`**`)`	finds and removes the given value from this list
`removeAll(`**`list`**`)`	removes any elements found in the given list from this list
`retainAll(`**`list`**`)`	removes any elements not found in given list from this list
`subList(`**`from, to`**`)`	returns the sub-portion of the list between indexes **from** (inclusive) and **to** (exclusive)
`toArray()`	returns the elements in this list as an array

Type Parameters (Generics)

```
ArrayList<Type> name = new ArrayList<Type>();
```

- When constructing an **ArrayList**, you must specify the type of elements it will contain between < and >.
  - This is called a type parameter or a generic class.
  - Allows the same **ArrayList** class to store lists of different types.

```
ArrayList<String> names = new ArrayList<String>();
names.add("John Smith");
names.add("Jerry West");
```

ArrayList vs. Array

- Construction

```
String[] names = new String[5];
```

```
ArrayList<String> list = new ArrayList<String>();
```

- Storing a value

```
names[0] = "Daniel";
```

```
list.add("Daniel");
```

- Retrieving a value

```
String s = names[0];
```

```
String s = list.get(0);
```

Conditionals

- Doing something to each value that starts with “B”

```
for (int i = 0; i < names.length; i++) {
    if (names[i].startsWith("B")) { ... }
}

for (int i = 0; i < list.size(); i++) {
    if (list.get(i).startsWith("B")) { ... }
}
```

- Seeing whether the value "Bret" is found

```
for (int i = 0; i < names.length; i++) {
if (names[i].equals("Bret")) { ... }
}

if (list.contains("Bret")) { ... }
```

ArrayList as a parameter

```
public static void name(ArrayList<Type> name) {
```

- Example:

```
// Removes all plural words from the given list.
public static void removePlural(ArrayList<String> list) {
    for (int i = 0; i < list.size(); i++) {
        String str = list.get(i);
        if (str.endsWith("s")) {
        list.remove(i);
        i--;
        }
    }
}
```

- You can also return a list:

```
public static ArrayList<Type> methodName(params) {
```

ArrayList of primitives?

- The type you specify when creating an **ArrayList** must be an object type; it cannot be a primitive type.

```
// illegal -- int cannot be a type parameter
ArrayList<int> list = new ArrayList<int>();
```

- But we can still use **ArrayList** with primitive types by using special classes called wrapper classes in their place.

```
// creates a list of ints
ArrayList<Integer> list = new ArrayList<Integer>();
```

We can make an Integer object out on int!

Wrapper classes

Primitive Type	Wrapper Type
int	Integer
double	Double
char	Character
boolean	Boolean

- A wrapper is an object whose sole purpose is to hold a primitive value.
- Once you construct the list, use it with primitives as normal:

```
ArrayList<Double> grades = new ArrayList<Double>();
grades.add(3.2);
grades.add(2.7);
...
double myGrade = grades.get(0);
```