CIS 351-Data Structure-ArrayList
Feb 6, 2020

Farzana Rahman

Syracuse University

ArrayLists and arrays

- A ArrayList is like an array of Objects
- Differences between arrays and ArrayLists:
  - An array is a fixed size, but a ArrayList expands as you add things to it
    - This means you don't need to know the size beforehand
  - Arrays can hold primitives or objects, but ArrayLists can only hold objects
    - However, autoboxing can make it *appear* that an ArrayList can hold primitives

Creating a ArrayList

- Specify, in angle brackets after the name, the type of object that the class will hold
- Examples:
  - `ArrayList<String> vec1 = new ArrayList<String>();`
  - `ArrayList<String> vec2 = new ArrayList<String>(10);`

Adding elements to a ArrayList

- boolean add(Object ***obj***)
  - Appends the object ***obj*** to the end of this ArrayList
  - With generics, the ***obj*** must be of the correct type, or you get a compile-time (syntax) error

- void add(int ***index***, Object ***element***)
  - Inserts the ***element*** at position ***index*** in this ArrayList
  - The ***index*** must be greater than or equal to zero and less than or equal to the number of elements in the ArrayList
  - With generics, the ***obj*** must be of the correct type, or you get a compile-time (syntax) error

Removing elements

- boolean remove(Object ***obj***)
  - Removes the first occurrence of ***obj*** from this ArrayList
  - Returns true if an element was removed
  - Uses equals to test if it has found the correct element
- void remove(int ***index***)
  - Removes the element at position ***index*** from this ArrayList
- void clear()
  - Removes all elements

Accessing with and without generics

- Object get(int ***index***)
  - Returns the component at position ***index***

- Using get :
  - ```
    ArrayList<String> myList = new ArrayList<String>();
    myList.add("Some string");
    String s = myList.get(0);
    ```
    ```

Searching a ArrayList

- boolean contains(Object ***element***)
  - Tests if ***element*** is a component of this ArrayList
  - Uses equals to test if it has found the correct element
- int indexOf(Object ***element***)
  - Returns the index of the first occurrence of ***element*** in this ArrayList
  - Uses equals to test if it has found the correct element
  - Returns -1 if ***element*** was not found in this ArrayList
- int lastIndexOf(Object ***element***)
  - Returns the index of the last occurrence of ***element*** in this ArrayList
  - Uses equals to test if it has found the correct element
  - Returns -1 if ***element*** was not found in this ArrayList

Getting information

- boolean isEmpty()
  - Returns true if this ArrayList has no elements
- int size()
  - Returns the number of elements currently in this ArrayList
- Object[ ] toArray()
  - Returns an array containing all the elements of this ArrayList in the correct order

Conclusion

- A ArrayList is like an array of Objects
- The advantage of a ArrayList is that you don't need to know beforehand how big to make it
- The disadvantage of a ArrayList is that you can't use the special syntax for arrays

Wrapper classes

- A Wrapper class is a class whose object wraps or contains a primitive data types.
- When we create an object to a wrapper class, it contains a field and in this field, we can store a primitive data types.
- In other words, we can wrap a primitive value into a wrapper class object.

Primitive Data Type	Wrapper Class
char	Character
byte	Byte
short	Short
long	Integer
float	Float
double	Double
boolean	Boolean

```
public class MyClass
{
    public static void main(String[] args)
    {
        Integer myInt = 5;
        Double myDouble = 5.99;
        Character myChar = 'A';

        System.out.println(myInt);
        System.out.println(myDouble);
        System.out.println(myChar);
    }
}
```

Following **methods** are used to get the value **associated** with the corresponding **wrapper object**: intValue(), byteValue(), shortValue(), longValue(), floatValue(), doubleValue(), charValue(), booleanValue()

```java
public class MyClass
{
    public static void main(String[] args)
    {
        Integer myInt = 5; Double myDouble = 5.99;
        Character myChar = 'A';
        System.out.println(myInt.intValue());
        System.out.println(myDouble.doubleValue());
        System.out.println(myChar.charValue());
    }
}
```

```java
public class MyClass
{
    public static void main(String[] args)
    {
        Integer myInt = 100;
        String myString = myInt.toString();
        System.out.println(myString.length());
    }
}
```

Autoboxing

- Automatic conversion of primitive types to the object of their corresponding wrapper classes is known as autoboxing.
  - conversion of int to Integer, long to Long, double to Double etc.

```java
// Java program to demonstrate Autoboxing

import java.util.ArrayList;
class Autoboxing
{
    public static void main(String[] args)
    {
        char ch = 'a';

        // Autoboxing- primitive to Character object conversion
        Character a = ch;

        ArrayList<Integer> arrayList = new ArrayList<Integer>();

        // Autoboxing because ArrayList stores only objects
        arrayList.add(25);

        // printing the values from object
        System.out.println(arrayList.get(0));
    }
}
```

Unboxing

- It is just the reverse process of autoboxing. Automatically converting an object of a wrapper class to its corresponding primitive type is known as unboxing.
  - conversion of Integer to int, Long to long, Double to double etc.

```java
// Java program to demonstrate Unboxing
import java.util.ArrayList;

class Unboxing
{
    public static void main(String[] args)
    {
        Character ch = 'a';

        // unboxing - Character object to primitive conversion
        char a = ch;

        ArrayList<Integer> arrayList = new ArrayList<Integer>();
        arrayList.add(24);

        // unboxing because get method returns an Integer object
        int num = arrayList.get(0);

        // printing the values from primitive data types
        System.out.println(num);
    }
}
```