Lecture 11: Exceptions, Iterators, Object Methods

9/21/2020

Lists and Sets in Java

Lists

- In lecture, we've built two types of lists: ALists and SLLists
  - Similar to Python lists

```
List61B<Integer> L = new AList<>();
L.addLast(5);
L.addLast(10);
L.addLast(15);
L.print();
```

- We built a list from scratch, but Java provides a built-in `List` interface and several implementations, eg `ArrayList`

```
java.util.List<Integer> L = new java.util.ArrayList<>();
L.add(5);
L.add(10);
L.add(15);
System.out.println(L);
```

Lists in Real Java Code

- By including "import java.util.List" and "import java.util.ArrayList", we can make our code more compact

```
import java.util.List;
import java.util.ArrayList;

List<Integer> L = new ArrayList<>();
```

Sets in Java and Python

- Another data structure is the set
  - Stores a set of values with no duplicates. Has no sense of order.

```
Set<String> S = new HashSet<>();
S.add("Tokyo");
S.add("Beijing");
```

```
S.add("Lagos");
S.add("Sao Paulo");
Sysetme.out.println(S.contains("Tokyo"));
```

ArraySet

- Today we're going to write our own set called `ArraySet`

Goals

- Goal 1: Create a class ArraySet with the following methods:
  - add(value): Add the value to the set if it is not already present
  - contains(value): Checks to see if ArraySet contains the key
  - size(): Returns the size of the array

ArraySet (Basic Implementation)

```
public class ArraySet<T> {
    private T[] items;
    private int size;

    public ArraySet() {
        items = (T[]) new Object[100];
        size = 0;
    }

    public boolean contains(T x) {
        for (int i = 0; i < size; i += 1) {
            if (e.equals(items[i])) {
                return true;
            }
        }
        return false;
    }

    public void add(T x) {
        if (contains(x)) {
            return;
        }
        items[size] = x;
        size += 1;
    }

    public int size() {
        return size;
    }
}
```

Exceptions

Exceptions

- Basic idea:
  - When something goes really wrong, break the normal flow of control
  - So far, we've only seen implicit exceptions

```
public class ArraySet<T> {

    ...

    public void add(T x) {
        if (x == null) {
            throw new IllegalArgumentException("You can't add null to an
ArraySet");
        }
        if (contains(x)) {
            return;
        }
        items[size] = x;
        size += 1;
    }

    ...

}
```

Explicit Exceptions

- We can also throw our own exceptions using the **throw** keyword
  - Can provide more informative message to a user
  - Can provide more information to code that "catches" the exception
- Arguably this is a bad exception
  - Our code now crashes when someone tries to add a null
  - Other fixes:
    - Ignore nulls
    - Fix `contains` so it doesn't crash when it tries to add null

```
public boolean contains(T x) {
    if (items[i] == null) {
        if (x == null) {
            return true;
        }
    }
    if (e.equals(items[i])) {
        return true;
    }
}
```

Iteration

The Enhanced For Loop

- Java allows us to iterate through Lists and Sets using a convenient shorthand syntax sometimes called the "foreach" or "enhanced for" loop
    - Doesn't work with our ArraySet

```
for (int i : javaset) {
    System.out.println(i);
}
```

How Iteration Really Works

- An alternate, uglier way to iterate through a List is to use the iterator() method

```
ArraySet<Integer> = new ArraySet<>();

Iterator<Integer> seer = aset.iterator();

while (seer.hasNext()) {
    int i = seer.next();
    System.out.println(i);
}
```

- To make this work, the Set interface needs to have an iterator() method and the iterator interface needs to have next/hasNext() methods

Support Ugly Iteration in ArraySets

- To support ugly iteration:
    - Add an `iterator()` method to `ArraySet` that returns an `Iterator<T>`
    - The `Iterator<T>` that we return should have a useful `hasNext()` and `next()` method

```
public class ArraySet<T> implements Iterable<T> {

    ...

    public Iterator<T> iterator() {
        return new ArraySetIterator();
    }

    private class ArraySetIterator implements Iterator<T> {
        private int wizPos;
        public ArraySetIterator() {
            wizPos = 0;
        }
```

```
        public boolean hasNext() {
            return wizPos < size;
        }

        public T next() {
            T returnItem = items[wizPos];
            wizPos += 1;
            return returnItem;
        }
    }
}
```

The Enhanced For Loop

- The problem: Java isn't smart enough to realize that our ArraySet has an iterator() method
  - Luckily there's an interface for that
- To support the enhanced for loop, we need to make `ArraySet` implement the `Iterable` interface

```
public interface Iterable<T> {
    Iterator<T> iterator();
}

public class ArraySet<T> implements Iterable<T> {

    ...

    public Iterator<T> iterator() {
        return new ArraySetIterator();
    }

    private class ArraySetIterator implements Iterator<T> {
        private int wizPos;
        public ArraySetIterator() {
            wizPos = 0;
        }

        public boolean hasNext() {
            return wizPos < size;
        }

        public T next() {
            T returnItem = items[wizPos];
            wizPos += 1;
            return returnItem;
        }
    }
}
```

The Iterable Interface

- By the way, this is how `Set` works as well
  - `Set` implements `Collection` which implements `Iterable`

Iteration Summary

- To support the enhanced for loop:
  - Add an `iterator()` method to your class that returns an `Iterator<T>`
  - The `Iterator<T>` returned should have a useful `hasNext()` and `next()` method
  - Add `implements Iterable<T>` to the line defining your class

Object Methods: Equals and toString()

Objects

- All classes are hyponyms of `Object`

toString()

- The `toString()` method provides a string representation of an object
  - `System.out.println(Object x)` calls `x.toString()`
    - println calls String.valueOf which calls toString
  - The implementation of toString() in Object is the name of the class, then an @ sign, then the memory location of the object

ArraySet toString

- One approach is shown below
  - Warning: This code is slow. Intuition: Adding even a single character to a string creates an entirely new string. Will discuss why at the end of the course.

```
public class ArraySet<T> implements Iterable<T> {

    ...

    @Override
    public String toString() {
        String returnString = "{";
        for (int i = 0; i < size - 1; i += 1) {
            returnString += item.toString();
            return String += ", ";
        }
        returnString += items[size - 1];
        returnString += "}";
        return returnString;
    }
}
```

ArrayMap toString

- Intuition: Append operation for a StringBuilder is fast

```
public class ArraySet<T> implements Iterable<T> {

    ...

    @Override
    public String toString() {
        StringBuilder returnSB = new StringBuilder("{");
        for (int i = 0; i < size - 1; i += 1) {
            returnSB.append(items[i].toString());
            returnSB.append(", ");
        }
        returnS.append(items[size - 1]);
        returnString.append("}");
        return returnSB.toString();
    }
}
```

Equals vs. ==

- As mentioned before, `==` and `.equals()` behave differently
  - `==` compares the bits. For references, `==` means "referencing the same object"

```
Set<Integer> javaset = Set.of(5, 23, 42);
Set<Integer> javaset2 = Set.of(5, 23, 42);
System.out.println(javaset == javaset2); // Prints false
```

- To test equality in the sense we usually mean it, use:
  - `.equals` for classes. Requires writing a `.equals` method for your own classes
    - Default implementation of `.equals` uses `==`
  - BTW: Use `Arrays.equal` of `Arrays.deepEquals` for arrays

```
Set<Integer> javaset = Set.of(5, 23, 42);
Set<Integer> javaset2 = Set.of(5, 23, 42);
System.out.println(javaset.equals(javaset2)); // Prints true
```

The Default Implementation of Equals

- The below implementation is a good start, but fails with `null` and objects not of type `ArraySet`

```
public class ArraySet<T> implements Iterable<T> {

    ...
```

```
    @Override
    public boolean equals(Object other) {
        ArraySet<T> o = (ArraySet<T>) other;
        if (o.size() != this.size()) {
            return false;
        }
        for (T item: this) {
            if (o.contains(item)) {
                return false;
            }
        }
        return true;
    }
}
```

- Better implementation below, but we can speed things up

```
public class ArraySet<T> implements Iterable<T> {

    ...

    @Override
    public boolean equals(Object other) {
        if (other == null) {
            return false;
        }
        if (other.getClass() != this.getClass()) {
            return false;
        }
        ArraySet<T> o = (ArraySet<T>) other;
        if (o.size() != this.size()) {
            return false;
        }
        for (T item: this) {
            if (o.contains(item)) {
                return false;
            }
        }
        return true;
    }
}
```

- Even faster implementation, pretty close to what a standard equals method looks like

```
public class ArraySet<T> implements Iterable<T> {

    ...

    @Override
```

```
    public boolean equals(Object other) {
        if (this == other) {
            return true;
        }
        if (other == null) {
            return false;
        }
        if (other.getClass() != this.getClass()) {
            return false;
        }
        ArraySet<T> o = (ArraySet<T>) other;
        if (o.size() != this.size()) {
            return false;
        }
        for (T item: this) {
            if (o.contains(item)) {
                return false;
            }
        }
        return true;
    }
}
```

Summary

Summary

- We built our own Array based Set implementation
- To make it more industrial strength we:
  - Added an exception if a user tried to add null to the set
    - There are other ways to deal with nulls. Our choice was arguably bad.
  - Added support for "ugly" then "nice" iteration
    - Ugly iteration: Creating a subclass with next and hasNext methods
    - Nice iteration: Declaring that ArraySet implements Iterable
  - Added a toString() method
    - Beware of String concatenation
  - Added an equals(Object) method
    - Make sure to deal with null and non-ArraySet arguments!
    - Used getClass to check the class of the passed object. Use sparingly.