Lecture 11 (Inheritance 4)

Iterators, Object Methods

CS61B, Spring 2024 @ UC Berkeley

Slides credit: Josh Hug

Today's Goal: ArraySet

Lecture 11, CS61B, Spring 2024

Today's Goal: ArraySet

Iteration

- The Enhanced For Loop
- iterator, next, hasNext
- iterator, next, hasNext for ArraySet
- Iterable

Object Methods

- == vs. equals
- toString
- Better toString (Bonus)
- .of (Bonus)

Sets in Java and Python

Today's goal: Build an implementation of a `Set` called `ArraySet`.

- Won't be implementing any `interface` (for now).
- Starting from basic implementation, we'll add some "industrial strength" features to the ArraySet like iteration, comparison, and toString.

```java
ArraySet<String> S = new ArraySet<>();
S.add("Oakland");
S.add("Toronto");
S.add("Minneapolis");
S.add("Oakland"); // no effect
S.add("Taipei");
System.out.println(S.contains("Oakland"));
```

```python
s = set()
s.add("Oakland")
s.add("Toronto")
s.add("Minneapolis")
s.add("Oakland") # no effect
s.add("Taipei")
print("Oakland" in s)
```

```
$ java SetExample
true
$ python set_example.py
True
```

Goals

Starting point: A class ArraySet with the following methods:

- `add(value)`: Add the value to the `ArraySet` if it is not already present.
- `contains(value)`: Checks to see if `ArraySet` contains the key.
- `size()`: Returns number of values.

For simplicity, I'll ignore resizing.

The basic functionality is quite straightforward, so I'll avoid live coding.

ArraySet (Basic Implementation)

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

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

"Unchecked cast" compiler warning here.
Nothing we can do about it.

Array implementation of a Set:

- Use an array as the core data structure.
- `contains(x)`: Checks to see if x is in the underlying array.
- `add(x)`: Checks to see if x is in the underlying array, and if not, adds it.

ArraySet (Basic Implementation)

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

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

We used `items[i].equals(x)`, not `items[i] == x`. This actually matters; more details later in this lecture.

Can also throw an IllegalArgumentException if you want to disallow null. See the videos for more details on exceptions.

The Enhanced For Loop

Lecture 11, CS61B, Spring 2024

Today's Goal: ArraySet

Iteration

- **The Enhanced For Loop**
- iterator, next, hasNext
- iterator, next, hasNext for ArraySet
- Iterable

Object Methods

- == vs. equals
- toString
- Better toString (Bonus)
- .of (Bonus)

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.

```
Set<Integer> javaset = new HashSet<>();
javaset.add(5);
javaset.add(23);
javaset.add(42);
for (int i : javaset) {
   System.out.println(i);
}
```

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.

- This doesn’t work with our ArraySet.
- Let’s strip away the magic so we can build our own classes that support this.

```
ArraySet<Integer> aset = new ArraySet<>();
aset.add(5);
aset.add(23);
aset.add(42);
for (int i : aset) {
   System.out.println(i);
}
```

```
$ javac IterationDemo
error: for-each not applicable to expression type
        for (int i : aset) {
                     ^
  required: array or java.lang.Iterable
  found:    ArraySet<Integer>
```

iterator, next, hasNext

Lecture 11, CS61B, Spring 2024

Today's Goal: ArraySet

Iteration

- The Enhanced For Loop
- **iterator, next, hasNext**
- iterator, next, hasNext for ArraySet
- Iterable

Object Methods

- == vs. equals
- toString
- Better toString (Bonus)
- .of (Bonus)

Why Doesn't the Enhanced For Loop Work?

The enhanced for loop works by first calling the `.iterator` method of the object.

- This returns an object of type `Iterator<Integer>`.
- The `Iterator` interface has its own API for fetching values one-by-one:
  - `hasNext`: Tells us whether there are more values.
  - `next`: gets the next value.

```
ArraySet<Integer> aset = new ArraySet<>();
aset.add(5);
aset.add(23);
aset.add(42);
for (int i : aset) {
   System.out.println(i);
}
```

What is missing?

How Iteration Really Works

An alternate, uglier way to iterate through a Set is to use the `iterator()` method.

Set.java:

```
public Iterator<E> iterator();
```

Suppose we have a **`Set<Integer>`** called **`javaset`**.

- In that case, we can iterate with either of the two equivalent pieces of code.
- Left code is shorthand for right code.

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

"Nice" iteration.

```
Iterator<Integer> seer
   = javaset.iterator();

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

"Ugly" iteration.

How Iterators Work

An alternate, uglier way to iterate through a Set is to use the `iterator()` method.

javaset:

5	23	42

```
Iterator<Integer> seer
   = javaset.iterator();
while (seer.hasNext()) {
   System.out.println(seer.next());
}
```

How Iterators Work

An alternate, uglier way to iterate through a Set is to use the `iterator()` method.

javaset:

5	23	42

```
$ java IteratorDemo.java
```

```
Iterator<Integer> seer
    = javaset.iterator();
while (seer.hasNext()) {
    System.out.println(seer.next());
}
```

How Iterators Work

An alternate, uglier way to iterate through a Set is to use the `iterator()` method.

javaset:

5	23	42

```
$ java IteratorDemo.java
```

```
Iterator<Integer> seer
    = javaset.iterator();
while (seer.hasNext()) {
    System.out.println(seer.next());
}
```

How Iterators Work

An alternate, uglier way to iterate through a Set is to use the `iterator()` method.

javaset:	5	23	42

```
$ java IteratorDemo.java
5
```

```
Iterator<Integer> seer
    = javaset.iterator();
while (seer.hasNext()) {
    System.out.println(seer.next());
}
```

How Iterators Work

An alternate, uglier way to iterate through a Set is to use the `iterator()` method.

javaset:	5	23	42

```
$ java IteratorDemo.java
5
```

```
Iterator<Integer> seer
    = javaset.iterator();
while (seer.hasNext()) {
    System.out.println(seer.next());
}
```

How Iterators Work

An alternate, uglier way to iterate through a Set is to use the `iterator()` method.

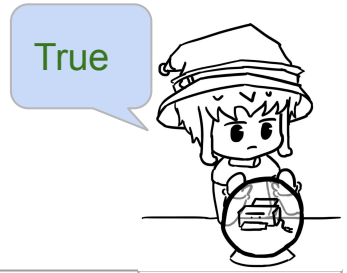
javaset:	5	23	42

```
$ java IteratorDemo.java
5
23
```

```
Iterator<Integer> seer
    = javaset.iterator();
while (seer.hasNext()) {
    System.out.println(seer.next());
}
```

How Iterators Work

An alternate, uglier way to iterate through a Set is to use the `iterator()` method.



javaset:

5	23	42

```
$ java IteratorDemo.java
5
23
```

```
Iterator<Integer> seer
    = javaset.iterator();
while (seer.hasNext()) {
    System.out.println(seer.next());
}
```

How Iterators Work

An alternate, uglier way to iterate through a Set is to use the `iterator()` method.

javaset:

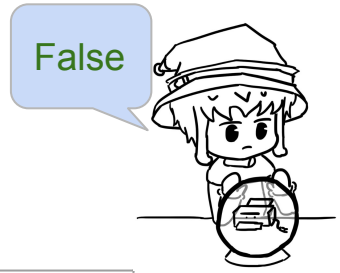
5	23	42

```
$ java IteratorDemo.java
5
23
42
```

```
Iterator<Integer> seer
    = javaset.iterator();
while (seer.hasNext()) {
    System.out.println(seer.next());
}
```

How Iterators Work

An alternate, uglier way to iterate through a Set is to use the `iterator()` method.



javaset:

5	23	42

```
$ java IteratorDemo.java
5
23
42
```

```
Iterator<Integer> seer
    = javaset.iterator();
while (seer.hasNext()) {
    System.out.println(seer.next());
}
```

iterator, next, hasNext for ArraySet

Lecture 11, CS61B, Spring 2024

Today's Goal: ArraySet

Iteration

- The Enhanced For Loop
- iterator, next, hasNext
- **iterator, next, hasNext for ArraySet**
- Iterable

Object Methods

- == vs. equals
- toString
- Better toString (Bonus)
- .of (Bonus)

The Secret of the Enhanced For Loop yellkey.com/TODO

The secret: The code on the left is just shorthand for the code on the right. For code on right to compile, which checks does the compiler need to do?

A. Does the Set interface have an iterator() method?
B. Does the Set interface have next/hasNext() methods?
C. Does the Iterator interface have an iterator method?
D. Does the Iterator interface have next/hasNext() methods?

```
Set<Integer> javaset = new HashSet<Integer>();
```

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

```
Iterator<Integer> seer
   = javaset.iterator();
while (seer.hasNext()) {
   System.out.println(seer.next());
}
```

The Secret of the Enhanced For Loop

The secret: The code on the left is just shorthand for the code on the right. For code on right to compile, which checks does the compiler need to do?

A. Does the Set interface have an iterator() method?

B. Does the Set interface have next/hasNext() methods?

C. Does the Iterator interface have an iterator method?

D. Does the Iterator interface have next/hasNext() methods?

```
Set<Integer> javaset = new HashSet<Integer>();
```

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

```
Iterator<Integer> seer
   = javaset.iterator();
while (seer.hasNext()) {
   System.out.println(seer.next());
}
```

Supporting 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.

`Iterator<T>`

```
public interface Iterator<T> {
   boolean hasNext();
   T next();
}
```

```
Iterator<Integer> aseer
        = aset.iterator();

while (aseer.hasNext()) {
   System.out.println(aseer.next());
}
```

Coding Demo: Iteration

ArraySet.java

```
public class ArraySet<T> {
   public static void main(String[] args) {
       ArraySet<Integer> aset = new ArraySet<>();
       aset.add(5);
       aset.add(23);
       aset.add(42);









   }
}
```

Coding Demo: Iteration

ArraySet.java

```
public class ArraySet<T> {
   public static void main(String[] args) {
       ArraySet<Integer> aset = new ArraySet<>();
       aset.add(5);
       aset.add(23);
       aset.add(42);

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









   }
}
```

Coding Demo: Iteration

ArraySet.java

```
public class ArraySet<T> {
   public static void main(String[] args) {
       ArraySet<Integer> aset = new ArraySet<>();
       aset.add(5);
       aset.add(23);
       aset.add(42);

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

       while (aseer.hasNext()) {


       }




   }
}
```

Coding Demo: Iteration

ArraySet.java

```
public class ArraySet<T> {
   public static void main(String[] args) {
       ArraySet<Integer> aset = new ArraySet<>();
       aset.add(5);
       aset.add(23);
       aset.add(42);

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

       while (aseer.hasNext()) {
           int i = aseer.next();

       }




   }
}
```

Coding Demo: Iteration

ArraySet.java

```
public class ArraySet<T> {
   public static void main(String[] args) {
       ArraySet<Integer> aset = new ArraySet<>();
       aset.add(5);
       aset.add(23);
       aset.add(42);

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

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









   }
}
```

Coding Demo: Iteration

ArraySet.java

```
public class ArraySet<T> {

   public Iterator<T> iterator() {

   }









}
```

Coding Demo: Iteration

ArraySet.java

```
public class ArraySet<T> {
   /** returns an iterator (a.k.a. seer) into ME */
   public Iterator<T> iterator() {

   }




}
```

Coding Demo: Iteration

ArraySet.java

```
public class ArraySet<T> {
   /** returns an iterator (a.k.a. seer) into ME */
   public Iterator<T> iterator() {
       return new ???;
   }

}
```

Coding Demo: Iteration

ArraySet.java

```
public class ArraySet<T> {
   /** returns an iterator (a.k.a. seer) into ME */
   public Iterator<T> iterator() {
       return new ???;
   }

   private class ArraySetIterator implements Iterator<T> {









   }
}
```

Coding Demo: Iteration

ArraySet.java

```
public class ArraySet<T> {
   /** returns an iterator (a.k.a. seer) into ME */
   public Iterator<T> iterator() {
       return new ???;
   }

   private class ArraySetIterator implements Iterator<T> {









       public boolean hasNext() {

       }









   }
}
```

Coding Demo: Iteration

ArraySet.java

```
public class ArraySet<T> {
   /** returns an iterator (a.k.a. seer) into ME */
   public Iterator<T> iterator() {
       return new ???;
   }

   private class ArraySetIterator implements Iterator<T> {









       public boolean hasNext() {

       }

       public T next() {



       }
   }
}
```

Coding Demo: Iteration

ArraySet.java

```
public class ArraySet<T> {
   /** returns an iterator (a.k.a. seer) into ME */
   public Iterator<T> iterator() {
       return new ArraySetIterator();
   }

   private class ArraySetIterator implements Iterator<T> {









       public boolean hasNext() {

       }

       public T next() {



       }
   }
}
```

Coding Demo: Iteration

ArraySet.java

```
public class ArraySet<T> {
   /** returns an iterator (a.k.a. seer) into ME */
   public Iterator<T> iterator() {
       return new ArraySetIterator();
   }

   private class ArraySetIterator implements Iterator<T> {
       private int wizPos;









       public boolean hasNext() {

       }

       public T next() {



       }
   }
}
```

Coding Demo: Iteration

ArraySet.java

```
public class ArraySet<T> {
   /** returns an iterator (a.k.a. seer) into ME */
   public Iterator<T> iterator() {
       return new ArraySetIterator();
   }

   private class ArraySetIterator implements Iterator<T> {
       private int wizPos;

       public ArraySetIterator() {

       }

       public boolean hasNext() {

       }

       public T next() {



       }
   }
}
```

Coding Demo: Iteration

ArraySet.java

```
public class ArraySet<T> {
   /** returns an iterator (a.k.a. seer) into ME */
   public Iterator<T> iterator() {
       return new ArraySetIterator();
   }

   private class ArraySetIterator implements Iterator<T> {
       private int wizPos;

       public ArraySetIterator() {
           wizPos = 0;
       }

       public boolean hasNext() {

       }

       public T next() {



       }
   }
}
```

Coding Demo: Iteration

ArraySet.java

```
public class ArraySet<T> {
   /** returns an iterator (a.k.a. seer) into ME */
   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() {



       }
   }
}
```

Coding Demo: Iteration

ArraySet.java

```
public class ArraySet<T> {
   /** returns an iterator (a.k.a. seer) into ME */
   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];



       }
   }
}
```

Coding Demo: Iteration

ArraySet.java

```
public class ArraySet<T> {
   /** returns an iterator (a.k.a. seer) into ME */
   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;

       }
   }
}
```

Coding Demo: Iteration

ArraySet.java

```
public class ArraySet<T> {
   /** returns an iterator (a.k.a. seer) into ME */
   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;
       }
   }
}
```

Completed ArraySet iterator Method

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.

```
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;
    }
}
```

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

Iterable

Lecture 11, CS61B, Spring 2024

Today's Goal: ArraySet

Iteration

- The Enhanced For Loop
- iterator, next, hasNext
- iterator, next, hasNext for ArraySet
- **Iterable**

Object Methods

- == vs. equals
- toString
- Better toString (Bonus)
- .of (Bonus)

The Enhanced For Loop

Our code now supports “ugly” iteration, but enhanced for loop still doesn’t work.

The problem: Java isn’t smart enough to realize that our `ArraySet` has an `iterator()` method.

- Luckily there’s an interface for that.

```
ArraySet<Integer> aset = new ArraySet<>();
aset.add(5);
aset.add(23);
aset.add(42);
for (int i : aset) {
   System.out.println(i);
}
```

```
$ javac IterationDemo
error: for-each not applicable to expression type
        for (int i : aset) {
                     ^
  required: array or java.lang.Iterable
  found:    ArraySet<Integer>
```

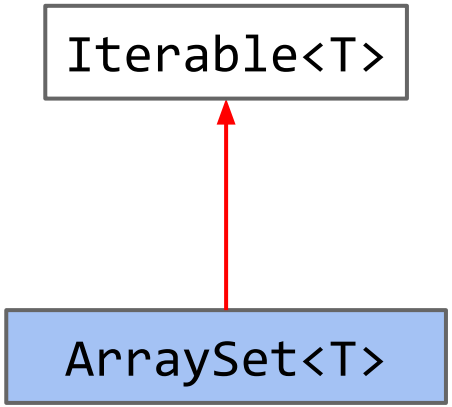
For-each Iteration And ArraySets

To support the enhanced for loop, we need to make `ArraySet` implement the `Iterable` interface.

- There are also some default methods in Iterable, not shown.

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

```
public class ArraySet<T> implements Iterable<T> {
   ...
   public Iterator<T> iterator() { ... }
}
```



Coding Demo: Iteration

ArraySet.java

```
public class ArraySet<T> {
   /** returns an iterator (a.k.a. seer) into ME */
   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;
       }
   }
}
```

Coding Demo: Iteration

ArraySet.java

```
public class ArraySet<T> implements Iterable<T> {
   /** returns an iterator (a.k.a. seer) into ME */
   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

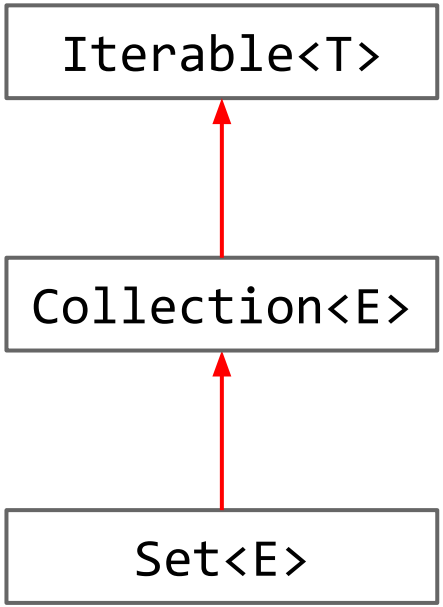
By the way, this is how `Set` works as well.

- Source code for Iterable: Link, Set: Link, Collection: Link.

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

```
public interface Collection<E> extends Iterable<E> {
   public Iterator<E> iterator();
}
```

```
public interface Set<E> extends Collection<E> {
   public Iterator<E> iterator();
}
```



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.

We'll do this in project 1C.

toString

Lecture 11, CS61B, Spring 2024

Today's Goal: ArraySet

Iteration

- The Enhanced For Loop
- iterator, next, hasNext
- iterator, next, hasNext for ArraySet
- Iterable

Object Methods

- **toString**
- == vs. equals
- Better toString (Bonus)
- .of (Bonus)

Object Methods

All classes are hyponyms of `Object`.

- **`String toString()`**
- `boolean equals(Object obj)`
- `int hashCode()`
- `Class<?> getClass()`
- `protected Object clone()`
- `protected void finalize()`
- `void notify()`
- `void notifyAll()`
- `void wait()`
- `void wait(long timeout)`
- `void wait(long timeout, int nanos)`

Today

Coming later.

Won't discuss or use in 61B.

toString()

The `toString()` method provides a string representation of an object.

- `System.out.println(Object x)` calls `x.toString()`
  - If you're curious: println calls String.valueOf which calls toString

```
Set<Integer> javaset = new HashSet<>();
javaset.add(5);
javaset.add(23);
javaset.add(42);

System.out.println(javaset);
```

```
$ java JavaSetPrintDemo
[5, 23, 42]
```

toString()

The `toString()` method provides a string representation of an object.

- `System.out.println(Object x)` calls `x.toString()`
- The implementation of toString() in Object is the the name of the class, then an @ sign, then the memory location of the object.
  - See 61C for what the “memory location” really means.

```
ArraySet<Integer> aset = new ArraySet<>();
aset.add(5);
aset.add(23);
aset.add(42);

System.out.println(aset);
```

```
$ java ArraySetPrintDemo
ArraySet@75412c2f
```

ArraySet toString

Let’s try implementing toString for ArraySet.

Coding Demo: toString

ArraySet.java

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

   public String toString() {









   }
}
```

Coding Demo: toString

ArraySet.java

```
public class ArraySet<T> implements Iterable<T> {
    @Override
    public String toString() {









    }
}
```

Coding Demo: toString

ArraySet.java

```
public class ArraySet<T> implements Iterable<T> {
    @Override
    public String toString() {
        String returnString = "{";









    }
}
```

Coding Demo: toString

ArraySet.java

```
public class ArraySet<T> implements Iterable<T> {
    @Override
    public String toString() {
        String returnString = "{";
        for (T item : this) {


        }









    }
}
```

Coding Demo: toString

ArraySet.java

```
public class ArraySet<T> implements Iterable<T> {
    @Override
    public String toString() {
        String returnString = "{";
        for (T item : this) {
            returnString += item.toString();

        }









    }
}
```

Coding Demo: toString

ArraySet.java

```
public class ArraySet<T> implements Iterable<T> {
    @Override
    public String toString() {
        String returnString = "{";
        for (T item : this) {
            returnString += item.toString();
            returnString += ", ";
        }









    }
}
```

Coding Demo: toString

ArraySet.java

```
public class ArraySet<T> implements Iterable<T> {
    @Override
    public String toString() {
        String returnString = "{";
        for (T item : this) {
            returnString += item.toString();
            returnString += ", ";
        }
        returnString += "}";









    }
}
```

Coding Demo: toString

ArraySet.java

```
public class ArraySet<T> implements Iterable<T> {
    @Override
    public String toString() {
        String returnString = "{";
        for (T item : this) {
            returnString += item.toString();
            returnString += ", ";
        }
        returnString += "}";
        return returnString;









    }
}
```

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 end of course.

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

Spoiler: It's because Strings are "immutable".

`keys[i]` might not be a string, but Java will automatically call `toString` so that you can add it to a string.

You can modify this code to avoid the extra comma at the end, if you want.

ArraySet toString

Much faster approach is shown below.

- Intuition: Append operation for a StringBuilder is fast.
- See the videos for more details about StringBuilder.

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

== vs. equals

Lecture 11, CS61B, Spring 2024

Today's Goal: ArraySet

Iteration

- The Enhanced For Loop
- iterator, next, hasNext
- iterator, next, hasNext for ArraySet
- Iterable

Object Methods

- toString
- **== vs. equals**
- Better toString (Bonus)
- .of (Bonus)

Object Methods

All classes are hyponyms of `Object`.

- `String toString()`
- **`boolean equals(Object obj)`**
- `int hashCode()`
- `Class<?> getClass()`
- `protected Object clone()`
- `protected void finalize()`
- `void notify()`
- `void notifyAll()`
- `void wait()`
- `void wait(long timeout)`
- `void wait(long timeout, int nanos)`

Coming in another lecture soon.

Coming later.

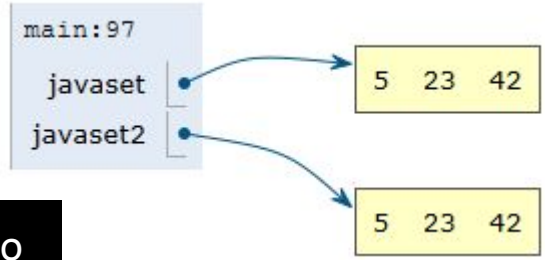
Won't discuss or use in 61B.

Equals vs. ==

As mentioned in an offhand manner previously, == 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);
```



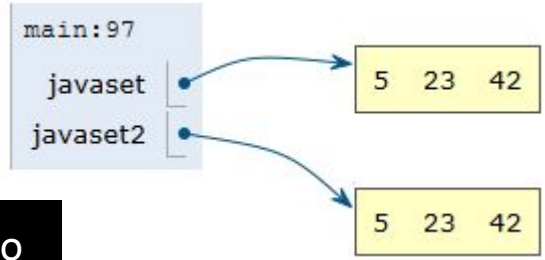

```
$ java EqualsDemo
False
```

Equals vs. ==

As mentioned in an offhand manner previously, == 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.equals(javaset2));
```




```
$ java EqualsDemo
True
```

To test equality in the sense we usually mean it, use:

- `.equals` for classes. Requires writing a `.equals` method for your classes.
  - Default implementation of `.equals` uses == (probably not what you want).
- BTW: Use `Arrays.equal` or `Arrays.deepEquals` for arrays.

`this`: Address of Current Object

`this` is a reference to the current object. Example from lecture 2:

```
public Dog maxDog(Dog uddaDog) {
    if (size > uddaDog.size) {
        return this;
    }
    return uddaDog;
}
```

this: Address of Current Object

Naturally, can also use this to access your own instance variables or methods.

- Unlike Python, where self is mandatory, using this is not mandatory.
- Two code snippets below are exactly identical in behavior.

```
public Dog maxDog(Dog o) {
   if (this.size > o.size) {
       return this;
   }
   return o;
}
```

```
public Dog maxDog(Dog o) {
   if (size > o.size) {
       return this;
   }
   return o;
}
```

this

Naturally, can also use `this` to access your own instance variables or methods.

- Unlike Python, where `self` is mandatory, using `this` is not mandatory.
- If there's ever a name conflict where a local variable has the same name as an instance variable (left), you **must** use `this` if you want to access the instance variable.

```
public Dog(int size) {
   size = size;
}
```

Does nothing!

```
public Dog(int s) {
   size = s;
}
```

Works correctly!

```
public Dog(int size) {
   this.size = size;
}
```

Works correctly!

```
public Dog(int s) {
   this.size = s;
}
```

Works correctly!

The Default Implementation of Equals

Below, we see the actual code for the default equals method in Object.java.

- Code available here if you want to poke around: https://github.com/openjdk/jdk17/blob/master/src/java.base/share/classes/java/lang/Object.java#L162

```
public class Object {

    ...

    public boolean equals(Object obj) {
        return (this == obj);
    }
}
```

The Default Implementation of Equals

```
ArraySet<Integer> aset = new ArraySet<>();
aset.add(5);
aset.add(23);
aset.add(42);

System.out.println(aset);

ArraySet<Integer> aset2 = new ArraySet<>();
aset2.add(5);
aset2.add(23);
aset2.add(42);

System.out.println(aset.equals(aset2));
```

```
$ java EqualsDemo
False
```

Returns false because the default implementation of equals just uses ==.

instanceOf Demo

The `instanceof` keyword is very powerful in Java.

- Checks to see if o's dynamic type is `Dog` (or one of its subtypes).
  If no, returns false.
- If yes, returns true *and* casts `o` into a variable of static type `Dog` called `uddaDog`.
- Works correctly, even if `o` is `null`.

```
@Override
public boolean equals(Object o) {
    if (o instanceof Dog uddaDog) {
        return this.size == uddaDog.size;
    }
    return false;
}
```

Let's try to write `ArraySet`'s `equals` method.

Coding Demo: equals

ArraySet.java

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

   public boolean equals(ArraySet o) {









   }
}
```

Coding Demo: equals

ArraySet.java

```
public class ArraySet<T> implements Iterable<T> {
   @Override
   public boolean equals(ArraySet o) {









   }
}
```

Compiler error. Not actually overriding the equals method in the Object class.

Coding Demo: equals

ArraySet.java

```
public class ArraySet<T> implements Iterable<T> {
   @Override
   public boolean equals(Object o) {









   }
}
```

Coding Demo: equals

ArraySet.java

```
public class ArraySet<T> implements Iterable<T> {
   @Override
   public boolean equals(Object o) {
       if (o instanceof ArraySet oas) {









       }



   }
}
```

Coding Demo: equals

ArraySet.java

```
public class ArraySet<T> implements Iterable<T> {
   @Override
   public boolean equals(Object o) {
       if (o instanceof ArraySet oas) {









       }
       // o is not an arrayset, so return false
       return false;


   }
}
```

Coding Demo: equals

ArraySet.java

```
public class ArraySet<T> implements Iterable<T> {
   @Override
   public boolean equals(Object o) {
       if (o instanceof ArraySet oas) {
           // check sets are of the same size
           if (oas.size != this.size) {
               return false;
           }









       }
       // o is not an arrayset, so return false
       return false;


   }
}
```

Coding Demo: equals

ArraySet.java

```
public class ArraySet<T> implements Iterable<T> {
   @Override
   public boolean equals(Object o) {
       if (o instanceof ArraySet oas) {
           // check sets are of the same size
           if (oas.size != this.size) {
               return false;
           }

           // check that all of MY items are in the other array set









       }
       // o is not an arrayset, so return false
       return false;


   }
}
```

Coding Demo: equals

ArraySet.java

```
public class ArraySet<T> implements Iterable<T> {
   @Override
   public boolean equals(Object o) {
       if (o instanceof ArraySet oas) {
           // check sets are of the same size
           if (oas.size != this.size) {
               return false;
           }

           // check that all of MY items are in the other array set
           for (T x : this) {



           }


       }
       // o is not an arrayset, so return false
       return false;


   }
}
```

Coding Demo: equals

ArraySet.java

```
public class ArraySet<T> implements Iterable<T> {
   @Override
   public boolean equals(Object o) {
       if (o instanceof ArraySet oas) {
           // check sets are of the same size
           if (oas.size != this.size) {
               return false;
           }

           // check that all of MY items are in the other array set
           for (T x : this) {
               if (!oas.contains(x)) {
                   return false;
               }
           }


       }
       // o is not an arrayset, so return false
       return false;


   }
}
```

Coding Demo: equals

ArraySet.java

```
public class ArraySet<T> implements Iterable<T> {
   @Override
   public boolean equals(Object o) {
       if (o instanceof ArraySet oas) {
           // check sets are of the same size
           if (oas.size != this.size) {
               return false;
           }

           // check that all of MY items are in the other array set
           for (T x : this) {
               if (!oas.contains(x)) {
                   return false;
               }
           }

           return true;
       }
       // o is not an arrayset, so return false
       return false;


   }
}
```

ArraySet equals

The code below is pretty close to what a standard equals method looks like.

```
@Override
public boolean equals(Object other) {
    if (this == other) { return true; }

    if (other instanceof ArraySet otherSet) {
        if (this.size != otherSet.size) { return false; }
        for (T x : this) {
            if (!otherSet.contains(x)) {
                return false;
            }
        }
        return true;
    }
    return false;
}
```

Technically a raw type without a type placeholder like `ArraySet<T>`, but don't worry about it.

ArraySet equals

The code below is pretty close to what a standard equals method looks like.

```
@Override
public boolean equals(Object other) {
    if (this == other) { return true; }

    if (other instanceof ArraySet otherSet) {
        if (this.size != otherSet.size) { return false; }
        for (T x : this) {
            if (!otherSet.contains(x)) {
                return false;
            }
        }
        return true;
    }
    return false;
}
```

Doesn't affect correctness, but saves us time if `this` and `other` reference the same object.

Historical Note: Old School Equals Methods

Equals methods written before March 2021 were ugly.

- Lots of manual type checking, casting, and null checking.
- See the CS61B 2021 slides.
- Some of TAs and AIs were taught this old way. You should avoid the old way (explicit casting). I prefer the new instanceof way.

```
@Override  // OLD SCHOOL APPROACH. NOT PREFERRED IN 61B.
public boolean equals(Object o) {
   if (o == null) { return false; }
   if (this == o) { return true; } // optimization
   if (this.getClass() != o.getClass()) { return false; }
   ArraySet<T> other = (ArraySet<T>) o;
   ...
}
```

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. (See videos.)
  - 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.
  - Used instanceof to check the class of the passed object.

Better toString (Bonus)

Lecture 11, CS61B, Spring 2024

Today's Goal: ArraySet

Iteration

- The Enhanced For Loop
- iterator, next, hasNext
- iterator, next, hasNext for ArraySet
- Iterable

Object Methods

- == vs. equals
- toString
- **Better toString (Bonus)**
- .of (Bonus)

The Lazy Way

Can use the `String.join` method to convert list of strings into a single string.

```
@Override
public String toString() {
    List<String> listOfItems = new ArrayList<>();
    for (T x : this) {
        listOfItems.add(x.toString());
    }
    return "{" + String.join(", ", listOfItems) + "}";
}
```

ArraySet.of (Bonus)

Lecture 11, CS61B, Spring 2024

Today's Goal: ArraySet Iteration

- The Enhanced For Loop
- iterator, next, hasNext
- iterator, next, hasNext for ArraySet
- Iterable

Object Methods

- == vs. equals
- toString
- Better toString (Bonus)
- **.of (Bonus)**

The of Method

We can write our own of method as follows:

- Below stuff is a so-called “var arg”.
  - Object passed is an array.
  - Values filled out using comma separated syntax.

```
public static <Glerp> ArraySet<Glerp> of(Glerp... stuff) {
    ArraySet<Glerp> returnSet = new ArraySet<Glerp>();
    for (Glerp x : stuff) {
        returnSet.add(x);
    }
    return returnSet;
}
```