Records, Arrays, Comparables, and Iterators

Records

- grouped data objects that are related as parts of a whole
- each field can be of different data type

```
public class Person {
    private String firstName;
    private String lastName;
    private String phoneNumber;
    private static int count = 0;
.
.
.
}
```

```
public class Person {
.
.
.
    public Person (String fn, String ln, String pn)
      throws Exception {
        this.firstName = fn;
        this.lastName = ln;
        this.phoneNumber = pn;
        this.validatePhoneNumber();
        Person.count++;
    }
.
.
.
}
```

```
public class Person {
.
.
.
    public Person (String name, String pn) throws Exception {
        String errorMsg = "Name format is xxxxx xxxxxxxx";
        String[] namePart = name.split(" ");
        if (namePart.length != 2)
            throw new Exception(errorMsg);
        this.firstName = namePart[0];
        this.lastName = namePart[1];
        this.phoneNumber = pn;
        this.validatePhoneNumber();
        Person.count++;
    }
.
.
.
}
```

```
public class Person {
.
.
.
    public String getName () {
        return this.firstName + " " + this.lastName;
    }

    public String getPhoneNumber () {
        return this.phoneNumber;
    }

    public static int getCount () {
        return Person.count;
    }
.
.
.
}
```

```
public class Person {
.
.
.
    private void validatePhoneNumber () throws Exception {
       String errorMsg = "Phone numbers must be xxx-xxx-xxxx";
       int i;
       if (!(this.phoneNumber.length() == 12) ||
           !Person.allDigits(this.phoneNumber.substring(0,3)) ||
           !Person.allDigits(this.phoneNumber.substring(4,7)) ||
           !Person.allDigits(this.phoneNumber.substring(8,12)) ||
           this.phoneNumber.charAt(3) != '-' ||
           this.phoneNumber.charAt(7) != '-')
         throw new Exception(errorMsg);
    }
.
.
.
}
```

```
public class Person {
.
.
.
    private static boolean allDigits (String s) {
        char c;
        for (int i = 0; i < s.length(); i++) {
            c = s.charAt(i);
            if (c < '0' || c > '9') return false;
        }
        return true;
    }
.
.
.
}
```

Arrays

- collection of data objects
- all data objects are of same type
- number of objects is fixed
- random access to objects in constant time
- arrays built in to Java
- Object[] a; allocates a single pointer to a Object[] initialized to null
- a = new Object[100]; then allocates 100 pointers to Objects, and assigns the address of the array to a.

```
public interface Set {
    public int size ();
    public boolean isEmpty ();
    public boolean isMember (Object e);
    public void add (Object e);
    public void remove (Object e);
    public Set union (Set that);
    public Set intersection (Set that);
    public Iterator iterator ();
    public Set copy ();
    public Set empty ();
}
```

```
public abstract class AbstractSet implements Set {
    public int size ()
        { ... }                         // O(n)
    public boolean isEmpty ()
        { ... }                         // O(size)
    public boolean isMember (Object e)
        { ... }                         // O(n)
    public Set union (Set that)
        { ... }                         // O(copy) + O(n*O(add))
    public Set intersection (Set that)
        { ... }                         // O(n*O(isMember + add))
    public abstract void add (Object e);
    public abstract void remove (Object e);
    public abstract Iterator iterator ();
    public abstract Set copy ();
    public abstract Set empty ();
}
```

Comparable Interface

- interface for objects that have an ordering
- must implement method

```
public int compareTo(Object obj);
```

```
public class ArraySet extends AbstractSet implements Set {
    private Comparable[] values;
    private int count;
.
.
.
}
```

```
public class ArraySet extends AbstractSet implements Set {
.
.
.
    public ArraySet (int max) {
        this.values = new Comparable[max];
        this.count = 0;
    }
.
.
.
}
```

```
public class ArraySet extends AbstractSet implements Set {
.
.
.
    // overriding AbstractSet size() with O(1) implementation
    // also makes isEmpty() time complexity O(1)

    public int size () {
        return this.count;
    }
.
.
.
}
```

```
public class ArraySet extends AbstractSet implements Set {
.
.
.
    public void add (Object obj) {
        if (this.isFull())
           throw new
           ArrayIndexOutOfBoundsException("Array Full");
        Comparable d = (Comparable) obj;
        int i = this.findIndex(d,0,this.count);
        if (i >= this.count ||
            d.compareTo(this.values[i]) != 0) {
           this.shiftDown(i);
           this.values[i] = d;
        }
    }
.
.
.
}
```

```
public class ArraySet extends AbstractSet implements Set {
.
.
.
    public void remove (Object obj) {
        Comparable d = (Comparable) obj;
        int i = this.findIndex(d,0,this.count);
        if (d.compareTo(this.values[i]) == 0)
           this.shiftUp(i);
    }
.
.
.
}
```

```
public class ArraySet extends AbstractSet implements Set {
.
.
.
    public ArraySetIterator iterator () {
        return new ArraySetIterator(this);
    }
.
.
.
}
```

```
public class ArraySet extends AbstractSet implements Set {
.
.
.
    public ArraySet copy () {
        ArraySet newSet = new ArraySet(this.values.length);
        for (int i = 0; i < this.count; i++)
            newSet.values[i] = this.values[i];
        newSet.count = this.count;
        return newSet;
    }
.
.
.
}
```

```
public class ArraySet extends AbstractSet implements Set {
.
.
.
    public ArraySet empty () {
        ArraySet newSet = new ArraySet(this.values.length);
        return newSet;
    }
.
.
.
}
```

```
public class ArraySet extends AbstractSet implements Set {
.
.
.
    // overriding AbstractSet with O(log n) implementation

    public boolean isMember (Object obj) {
        Comparable d = (Comparable) obj;
        int i = this.findIndex(d,0,this.count);
        return d.compareTo(this.values[i]) == 0;
    }
.
.
.
}
```

```
public class ArraySet extends AbstractSet implements Set {
.
.
.
    private int findIndex (Comparable d, int top, int out) {
        if (top == out) return top;
        int mid = (top + out) / 2;
        if (d.compareTo(this.values[mid]) == 0) return mid;
        if (d.compareTo(this.values[mid]) > 0)
           return this.findIndex(d,mid+1,out);
        else
           return this.findIndex(d,top,mid);
    }
.
.
.
}
```

```
public class ArraySet extends AbstractSet implements Set {
.
.
.
    private void shiftDown (int i) {
        for (int p = this.count++; p >= i; p--)
            this.values[p+1] = this.values[p];
    }

    private void shiftUp (int i) {
        this.count--;
        for (int p = i; p < this.count; p++)
            this.values[p] = this.values[p+1];
    }
.
.
.
}
```

```
public class ArraySet extends AbstractSet implements Set {
.
.
.
    public boolean isFull () {
        return this.count == this.values.length;
    }

    public Comparable get (int i) {
        return this.values[i];
    }
.
.
.
}
```

Iterator Interface

- interface for iterating over a collection
- must implement methods for

```
public boolean hasNext();
public Object next();
```

```
public class ArraySetIterator implements Iterator {

    private int nextIndex;
    private ArraySet theArraySet;

    public ArraySetIterator (ArraySet a) {
        this.nextIndex = 0;
        this.theArraySet = a;
    }

    public boolean hasNext () {
        return this.nextIndex < this.theArraySet.size();
    }

    public Object next () {
        return this.theArraySet.get(this.nextIndex++);
    }
}
```