Java 2 Correctness

- Parameterizing an entire class increases the scope of <T>.
- ArrayList uses type parameters
- Comparable uses type parameters
- You should always bind the parameters

- A class is generic if it declares one or more type variables.
- A generic type can have more than one type parameter.
- Don't use raw types.
  - They are bad. Bad. BAd. BAD. BADDDDDD

```
public class ArrayLiv<T extends Number> {
        public int search(T[] a, T target) {
                . . .
        }
}
```

```
public class ArrayLib<T extends Comparable> { //Comparable is still a raw type//
        public int search(T[] a, T target) {
                . . .
        }
}
```

```
public interface Comparable<T> { . . .}
```

```
public class ArrayLib<T extends Comparable<T>> { //Restricts types//
        public int search(T[] a, T target) {
                . . .
        }
}
```

Therefore it is type safe

*****STRATEGY PATTERN******

- Allow an algorithm's behavior to be selected at runtime.
- Comparator and Comparable are similar in the fact that they are designed to impose total order.
- Comparator has one method known as compare(T o1, T o2)

```
Public class ArrayLib {
        public static <T> int search(T[] a, T target, Comparator<T> c) {
                int i = 0;
                while ((i < a.length) && (c.compare(a[i], target) != 0))
                        i++;
                . . .
        }
}
```

- make a class that compares books by title that implements the Comparator interface

- calling the generic method
- ArrayLib.<Book>search(a, target, new CompareBooksByTitle());

- Using Arrays.sort() with comparators

- List allows you to replace arrays
- Lab4 is about iterators

```
public <T> int search(List<T> a, T target) {
        int i = 0;
        while(( i < a.size()) && (!a.get(i).equals(target))
}
```

- This code doesn't require any particular class to be used as long as it has signed the List contract.

*********ITERATOR PATTERN**************

- Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.
- Iterator interface- hasNext() (a boolean), next() (a type variable E), remove() (a void) (Optional Operation).

```
public <T> int search(List<T> a, T target) {
        Iterator<T> its = a.iterator();
        int i = 0;
        while(( itr.hasNext()) && (!itr.next().equals(target))
        i++;
        if (i < a.size())
                return i;
        else
                return -1;
}
```

- The iterator method is part of the List interface.
- Do Lab 4, Use Iterators in A2.
- Iterator starts just before the first element.
- Don't call next twice because hasNext will return false.

Benefits of using an Iterator:
- Generality - Allows traversals regardless of implementation, **Including the methods that are available.** *Not every collection will have a "get by index" method.*
- Efficiency - Depending on how the collection is implemented, the iterator could provide much faster access than the get() method in the loop.