Searching an element in a Collection

The `binarySearch(List list, T key)` method searches the specified list for the specified object using the binary search algorithm. The elements added in the List must implement the Comparable interface, and the list must be sorted into ascending order before making this call. If it is not sorted, the results are undefined. If the list contains multiple elements equal to the specified object, there is no guarantee which one will be found.

If the elements added to our List do not implement the Comparable interface, then we can use another overloaded version of `binarySearch(List list, T key, Comparator comp)`, which takes a Comparator for the input as well. The list must be sorted into ascending order according to the specified comparator.

If the element is not present in the list, then this method returns a position where the element can be inserted.

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

public class CollectionsDemo {

    public static void main(String args[]) {
        List<Integer> list = new ArrayList<>();
        list.add(9);
        list.add(12);
        list.add(34);
        list.add(54);
        list.add(66);
        list.add(76);

        System.out.println("The minimum element in the List is: " + Collections.binarySearch(list, new In
    }
}

```

Copying a list into another list

The `copy(List dest, List src)` method is used to copy all the elements from a source list to a destination list. If the size of the destination list is smaller than the source list, then `IndexOutOfBoundsException` is thrown. After the operation, the index of each copied element in the destination list will be identical to its index in the source list.

Let's consider an example to understand this.

```
List dest = 10,20,30,40,50,60;
List src = 1,2,3;
Collections.copy(dest, src);
```

After doing the above operation, the `dest` list will become:

```
{1,2,3,40,50,60}
```

So the `copy()` method does not merge the elements of the two lists. It replaces the elements in the destination list from the elements in the source list.

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

public class CollectionsDemo {

    public static void main(String args[]) {
        List<Integer> list1 = new ArrayList<>();
        list1.add(9);
        list1.add(12);
        list1.add(34);
        list1.add(54);
        list1.add(66);
        list1.add(76);

        List<Integer> list2 = new ArrayList<>();
        list2.add(90);
        list2.add(12);
        list2.add(98);
        list2.add(43);

        Collections.copy(list1, list2);

        System.out.println(list1);
    }
}

```

Filling a list with a default value

The `fill(List list, Object obj)` method replaces all of the elements of the specified list with the specified element. This method is very useful if we want to reset our List to a default value.

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

public class ArrayListDemo {

    public static void main(String args[]) {
        List<Integer> list = new ArrayList<>();
        list.add(34);
        list.add(45);

        Collections.fill(list, 10);

        System.out.println(list);
    }
}

```