

```
package genericity;

/**
 * A container for a single item. No copy is made of the contained item, so the
 * client retains an alias to the inserted item.
 *
 * @convention value is null ==> isEmpty
 * @correspondence isEmpty ==> contains is empty <br />
 * !isEmpty ==> value in contains
 * @author paolo
 * @param <T>
 */
public class PlasticBox<T> implements Box<T> {

    /**
     * The contained item, if one exists. If the PlasticBox is empty, there is no
     * guarantee on value. In particular, it might or might not be null.
     */
    private T value;

    /**
     * True if and only if the PlasticBox is empty. When isEmpty is true, it
     * might not be safe to dereference value, as that field might be null.
     */
    private boolean isEmpty;

    /**
     * Initializes a PlasticBox to be empty.
     *
     * @ensures isEmpty
     */
    PlasticBox() {
        this.isEmpty = true;
    }

    /**
     * @ensures contains <==> (!isEmpty and target = value)
     * @see genericity.Box#contains(java.lang.Object)
     */
    public boolean contains(T target) {
        if (!this.isEmpty) {
            if (target.equals(this.value)) {
                return true;
            }
        }
        return false;
    }

    /**
     * @alters this
     * @ensures #isEmpty ==> (!isEmpty and value = item) <br />
     * !#isEmpty ==> value = #value
     * @see genericity.Box#insert(java.lang.Object)
     */
    public void insert(T item) {
        if (this.isEmpty) {
            this.isEmpty = false;
            this.value = item;
        }
    }

    /**
     * @requires !isEmpty
     * @alters this
     */
}
```

```
    * @ensures isEmpty and removeAny = #value
    * @see genericity.Box#removeAny()
    */
    public T removeAny() {
        assert (!this.isEmpty);
        this.isEmpty = true;
        return this.value;
    }

    /**
     * @ensures isEmpty ==> size = 0 <br />
     * !isEmpty ==> size = 1
     * @see genericity.Box#size()
     */
    public int size() {
        if (this.isEmpty) {
            return 0;
        }
        return 1;
    }
}
```