```
/**
 * 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
 */
public class PlasticBox<T> implements Box<T> {

        /**
         * The contained item, if one exists. If the PlasicBox 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() {
                isEmpty = true;
        }

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

        /**
         * @see Box#insert(java.lang.Object)
         */
        public void insert(T item) {
                if (isEmpty) {
                        isEmpty = false;
                        value = item;
                }
        }

        /**
         * @see Box#removeAny()
         */
        public T removeAny() {
                assert (!isEmpty);
                isEmpty = true;
                return value;
        }

        /**
         * @see Box#size()
```

```
	 */
	public int size() {
		if (isEmpty) {
			return 0;
		}
		return 1;
	}

}
```