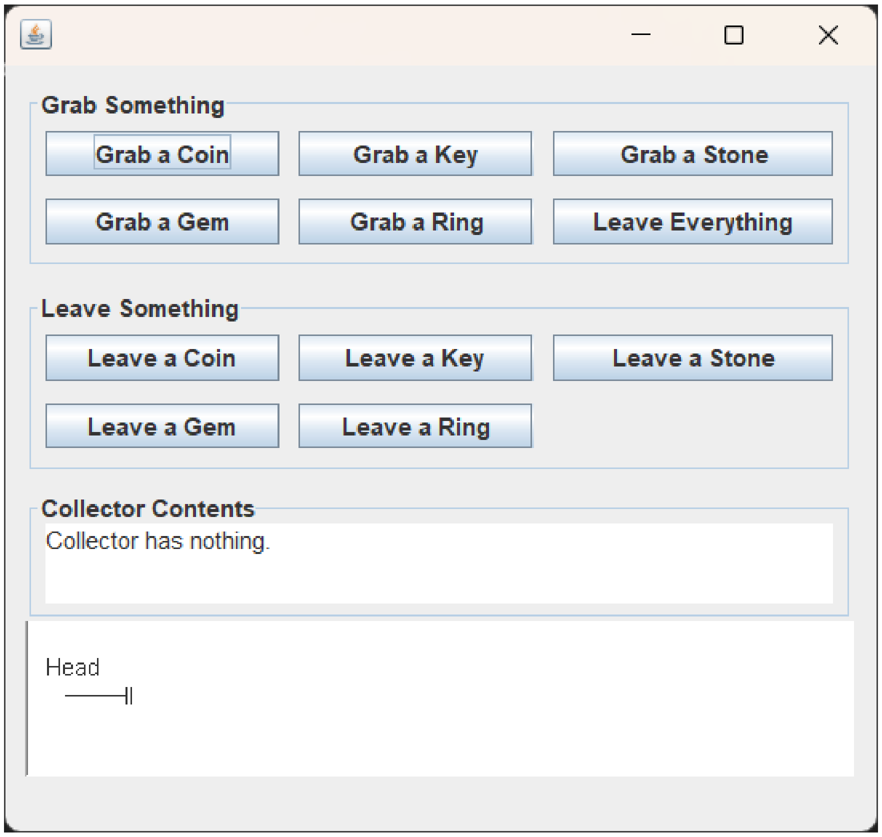
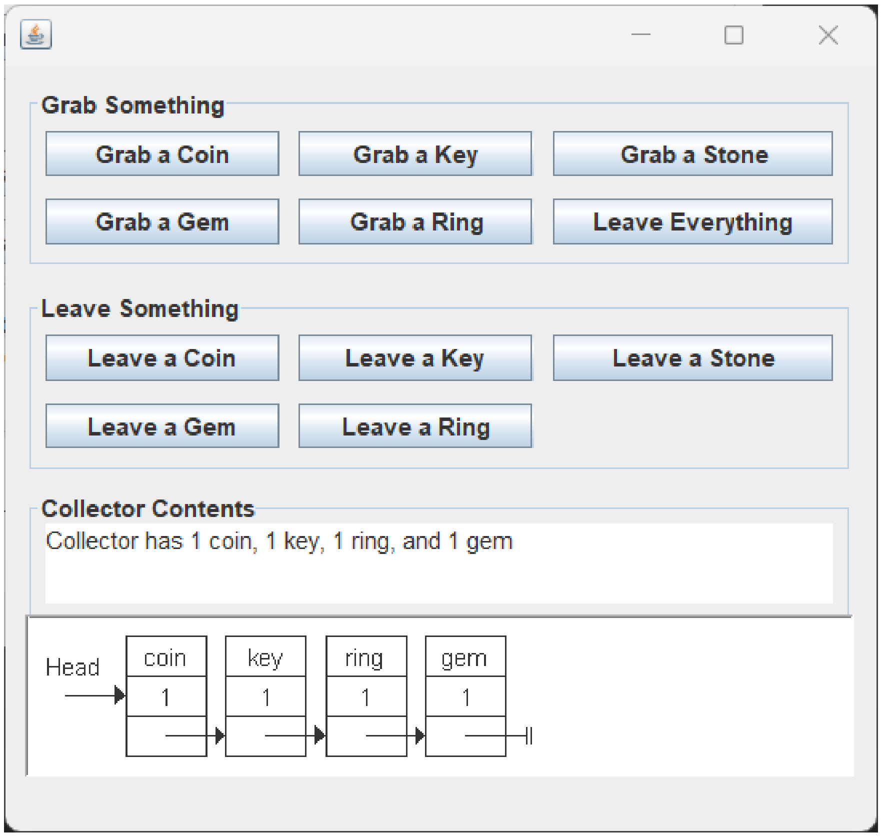
Empty node list



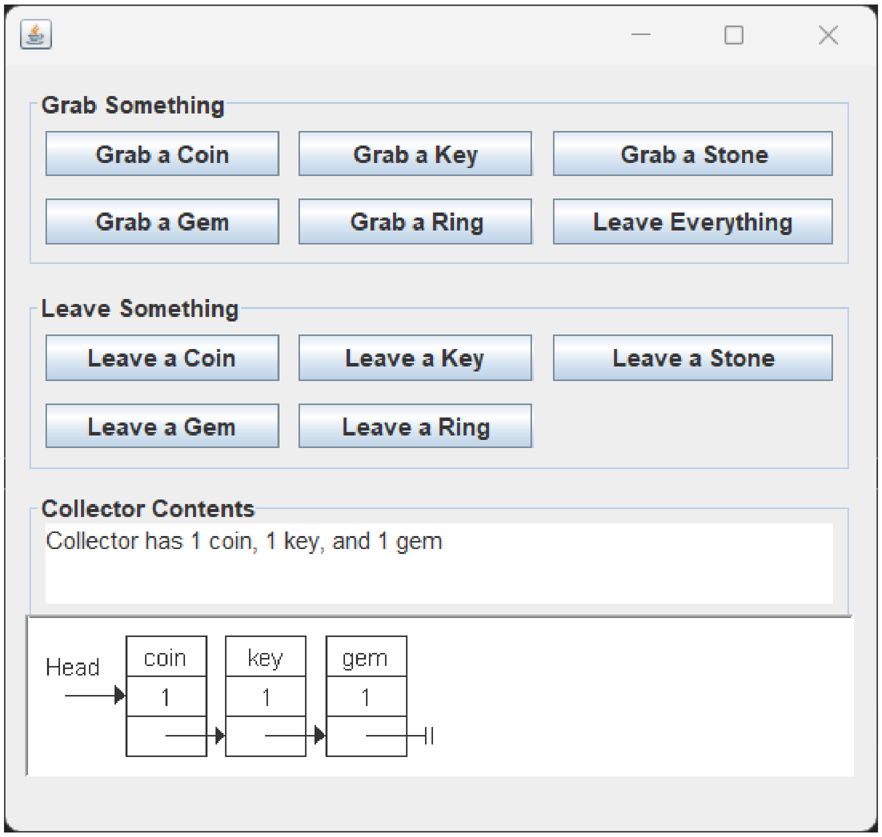
Collector has nothing.

Node list with a few example objects

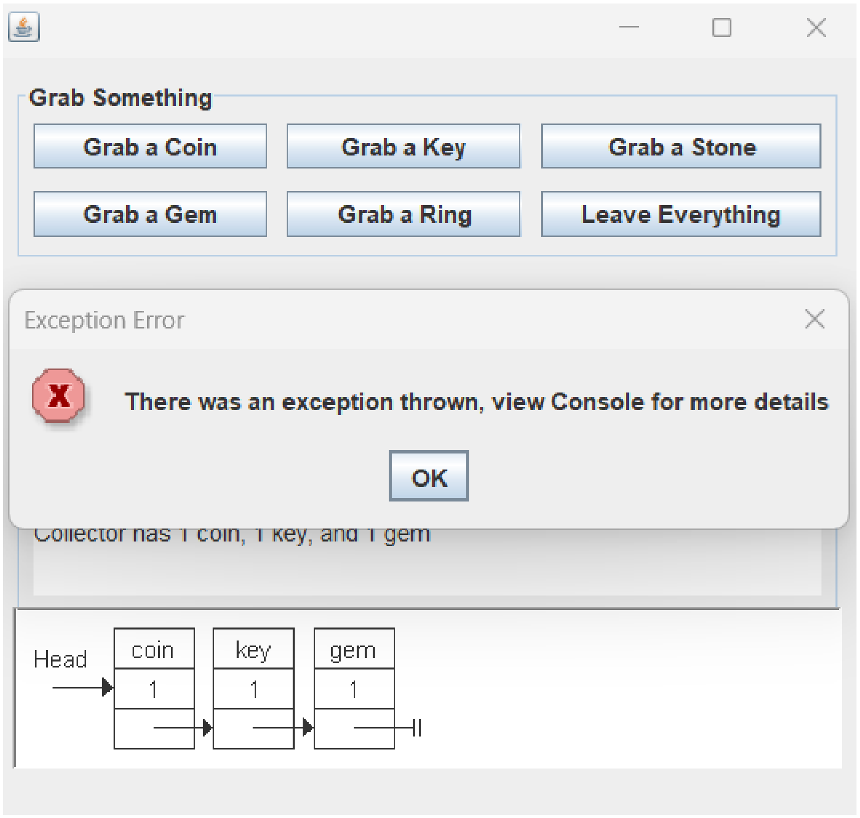


Collector has 1 coin, 1 key, 1 ring, and 1 gem

Node list after removing object and after attempting to remove nonexistent objects



Collector has 1 coin, 1 key, and 1 gem



There was an exception thrown, view Console for more details

Collector has 1 coin, 1 key, and 1 gem

Node logic:

```java
package edu.uwm.cs351;
public class Collector {
private final static String things[] = { "coin", "gem", "key", "ring", "stone" };
public enum ThingType {
NOTHING(-1), COIN(0), GEM(1), KEY(2), RING(3), STONE(4);
private int value;
private ThingType(int value) {
this.value = value;
}
public int getValue() {
return value;
}
public final static String describe(ThingType t) {
if (t == ThingType.NOTHING) {
return "nothing, check input!!";
}
else {
return things[t.getValue()];
}
}
}
private static class CollectorNode
{
ThingType thing;
int count;
CollectorNode next;
public CollectorNode (ThingType thing, int count, CollectorNode next) {
this.thing = thing;
this.count = count;
this.next = next;
}
@Override
public String toString () {
return thing.toString() + ": " + count;
}
}
private CollectorNode head;
public Collector() {
head = null;
}
/** Check if the collector is empty.
* @return true if collector is empty.
*/
public Boolean isEmpty() {
return head == null;
}
/** Counts the number of things in the collector.
* @return the number of things in the collector.
*/
public int howManyThings() {
int count = 0;
for (CollectorNode cur = head; cur != null; cur=cur.next) {
++count;
}
return count;
}
/** Count the number of particular kind of things.
* @param count_thing the type of thing to count.
* @return the number of that type.
*/
public int howManyOf(ThingType count_thing) {
for (CollectorNode cur = head; cur != null; cur=cur.next) {
```

```java
if (cur.thing == count_thing) return cur.count;
}
return 0;
}
/** Add one of this kind to the collector.
* @param new_thing the kind to add.
*/
public void grab(ThingType new_thing) {
// Being lazy is good. Never write two functions that are almost
// the same if one can be written to use the other, with
// change in function or in big-Oh efficiency.
grabSome(new_thing,1);
}
/**
* Grab something, if the item already exists in the collector increment the existing count
* by the new count
*
* @param new_thing, new_thing of ThingType to add
* @param count, number of
*/
public void grabSome(ThingType new_thing, int count) {
CollectorNode lag = null, p = null;
for(p= head; p!=null; lag=p, p=p.next) {
if (new_thing == p.thing) {
p.count += count;
return;
}
}
CollectorNode n = new CollectorNode(new_thing,count,p);
if (head == null) { head = n;}
else {lag.next = n;}
}
/** Remove one instance of lv_thing from this collector.
* @param lv_thing type of thing to remove.
*/
public void leave(ThingType lv_thing) {
CollectorNode lag= null, p;
if (head == null) {throw new IllegalStateException("No things to leave!");}
for (p=head;p!=null; lag=p, p=p.next) {
if(p.thing == lv_thing) {
if(p.count > 1) {p.count--;}
else {
//! --- Bug Section --- //
if (p== head) {head = head.next;}
else {lag.next = p.next;}
}
return;
}
}
throw new IllegalStateException("No matching thing to leave!");
}
/** Removes all of a particular type.
* @param lv_thing particular type.
*/
public void leaveAll(ThingType lv_thing) {
CollectorNode lag= null, p;
if (head == null) {throw new IllegalStateException("No things to leave!");}
for (p=head;p!=null; lag=p, p=p.next) {
if(p.thing == lv_thing) {
if (p== head) {head = head.next;}
else {lag.next = p.next;}
return;
}
}
throw new IllegalStateException("No matching thing to leave!");
```

```
}
/** Remove everything from the collector.
*/
public void leaveEverything() {
// while we have anything, drop all of what the first thing is.
while (!isEmpty()) {
leaveAll(whatIs(0));
}
}
/** Return the kind of thing at particular point in the linkedList.
* @param pos zero-based index.
* @return kind of thing at that position.
* @throws illegalStateException if position is out of range.
*/
public ThingType whatIs(int pos) {
if (pos < 0) {
throw new IllegalStateException("list what_is position negative");
}
CollectorNode cur = head;
for (int i = 0; i < pos; i++) {
if (cur == null) {
throw new IllegalStateException("list what_is position too large");
}
else {
cur = cur.next;
}
}
return cur.thing;
}
/** Return the position of this kind of thing in the linkedList.
* @param loc_thing kind of thing to look for.
* @return zero-based index.
* @throws illegalStateException if nothing of that type is present.
*/
public int whereIs(ThingType loc_thing) {
int i = 0;
CollectorNode cur = head;
while (cur != null && loc_thing != cur.thing) {
cur = cur.next;
i++;
}
if (cur == null) {
throw new IllegalStateException("Thing not found");
}
else return i;
}
/** Provide a explanation of what is in the collection.
* @return human-readable string.
*/
public String show() {
String s = "";
if (!this.isEmpty()) {
s += "Collector has ";
int how_many = howManyThings();
for (int j = 0; j < how_many; j++) {
ThingType t = this.whatIs(j);
int how_many_of_this = this.howManyOf(t);
s += how_many_of_this + " " + ThingType.describe(t);
if (how_many_of_this > 1) {
s += "s";
}
if (j < how_many - 1) {
s += ", ";
}
if (j == (how_many -2)) {
```

```
s += "and ";
}
}
}
else {
s = "Collector has nothing.";
}
return s;
}
}

```