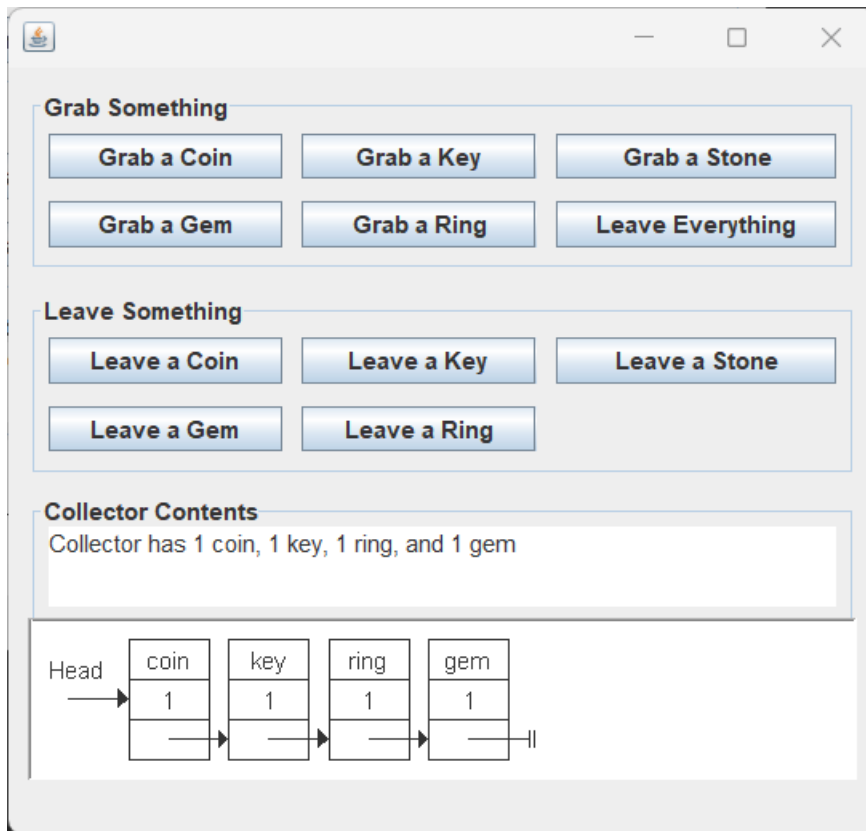


Empty node list

The window has a title bar with a standard Mac OS icon and window controls (minimize, maximize, close). The main content area is divided into three sections:

- Grab Something**: A section containing five buttons arranged in two rows. The first row has "Grab a Coin", "Grab a Key", and "Grab a Stone". The second row has "Grab a Gem", "Grab a Ring", and "Leave Everything".
- Leave Something**: A section containing four buttons arranged in two rows. The first row has "Leave a Coin", "Leave a Key", and "Leave a Stone". The second row has "Leave a Gem" and "Leave a Ring".
- Collector Contents**: A section containing a text area with the text "Collector has nothing." and a diagram below it. The diagram shows the word "Head" followed by a horizontal line and a vertical line.

Node list with a few example objects



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

The screenshot shows a Java Swing window with three sections:

- Grab Something:** Contains five buttons: "Grab a Coin", "Grab a Key", "Grab a Stone", "Grab a Gem", and "Leave Everything".
- Leave Something:** Contains three buttons: "Leave a Coin", "Leave a Key", and "Leave a Ring".
- Collector Contents:** A text area displaying "Collector has 1 coin, 1 key, and 1 gem".

Below the text area is a diagram of a linked list structure:

- A "Head" pointer points to the first node.
- The first node is a table with "coin" in the top row and "1" in the bottom row. Its "next" pointer points to the second node.
- The second node is a table with "key" in the top row and "1" in the bottom row. Its "next" pointer points to the third node.
- The third node is a table with "gem" in the top row and "1" in the bottom row. Its "next" pointer is null (represented by a vertical line).

The screenshot shows the same Java Swing window as above, but with an "Exception Error" dialog box overlaid. The dialog box contains:

- A red "X" icon.
- The text: "There was an exception thrown, view Console for more details".
- An "OK" button.

The background window is partially obscured by the dialog box, but the "Collector Contents" text area still shows "Collector has 1 coin, 1 key, and 1 gem". The linked list diagram is also visible below the dialog box.

Node logic:

```
1. package edu.uwm.cs351;
2. public class Collector {
3.     private final static String things[] = { "coin", "gem", "key", "ring", "stone" };
4.     public enum ThingType {
5.         NOTHING(-1), COIN(0), GEM(1), KEY(2), RING(3), STONE(4);
6.         private int value;
7.         private ThingType(int value) {
8.             this.value = value;
9.         }
10.    public int getValue() {
11.        return value;
12.    }
13.    public final static String describe(ThingType t) {
14.        if (t == ThingType.NOTHING) {
15.            return "nothing, check input!!";
16.        }
17.        else {
18.            return things[t.getValue()];
19.        }
20.    }
21. }
22. private static class CollectorNode
23. {
24.     ThingType thing;
25.     int count;
26.     CollectorNode next;
27.     public CollectorNode (ThingType thing, int count, CollectorNode next) {
28.         this.thing = thing;
29.         this.count = count;
30.         this.next = next;
31.     }
32.     @Override
33.     public String toString () {
34.         return thing.toString() + ": " + count;
35.     }
36. }
37. private CollectorNode head;
38. public Collector() {
39.     head = null;
40. }
41. /** Check if the collector is empty.
42.  * @return true if collector is empty.
43.  */
44. public Boolean isEmpty() {
45.     return head == null;
46. }
47. /** Counts the number of things in the collector.
48.  * @return the number of things in the collector.
49.  */
50. public int howManyThings() {
51.     int count = 0;
52.     for (CollectorNode cur = head; cur != null; cur=cur.next) {
53.         ++count;
54.     }
55.     return count;
56. }
57. /** Count the number of particular kind of things.
58.  * @param count_thing the type of thing to count.
59.  * @return the number of that type.
60.  */
61. public int howManyOf(ThingType count_thing) {
62.     for (CollectorNode cur = head; cur != null; cur=cur.next) {
```

```

63. if (cur.thing == count_thing) return cur.count;
64. }
65. return 0;
66. }
67. /** Add one of this kind to the collector.
68. * @param new_thing the kind to add.
69. */
70. public void grab(ThingType new_thing) {
71. // Being lazy is good. Never write two functions that are almost
72. // the same if one can be written to use the other, with
73. // change in function or in big-O efficiency.
74. grabSome(new_thing,1);
75. }
76. /**
77. * Grab something, if the item already exists in the collector increment the existing count
78. * by the new count
79. *
80. * @param new_thing, new_thing of ThingType to add
81. * @param count, number of
82. */
83. public void grabSome(ThingType new_thing, int count) {
84. CollectorNode lag = null, p = null;
85. for(p= head; p!=null; lag=p, p=p.next) {
86. if (new_thing == p.thing) {
87. p.count += count;
88. return;
89. }
90. }
91. CollectorNode n = new CollectorNode(new_thing,count,p);
92. if (head == null) { head = n;}
93. else {lag.next = n;}
94. }
95. /** Remove one instance of lv_thing from this collector.
96. * @param lv_thing type of thing to remove.
97. */
98. public void leave(ThingType lv_thing) {
99. CollectorNode lag= null, p;
100. if (head == null) {throw new IllegalStateException("No things to leave!");}
101. for (p=head;p!=null; lag=p, p=p.next) {
102. if(p.thing == lv_thing) {
103. if(p.count > 1) {p.count--;}
104. else {
105. ///! --- Bug Section --- //
106. if (p== head) {head = head.next;}
107. else {lag.next = p.next;}
108. }
109. return;
110. }
111. }
112. throw new IllegalStateException("No matching thing to leave!");
113. }
114. /** Removes all of a particular type.
115. * @param lv_thing particular type.
116. */
117. public void leaveAll(ThingType lv_thing) {
118. CollectorNode lag= null, p;
119. if (head == null) {throw new IllegalStateException("No things to leave!");}
120. for (p=head;p!=null; lag=p, p=p.next) {
121. if(p.thing == lv_thing) {
122. if (p== head) {head = head.next;}
123. else {lag.next = p.next;}
124. return;
125. }
126. }
127. throw new IllegalStateException("No matching thing to leave!");

```

```

128. }
129. /** Remove everything from the collector.
130. */
131. public void leaveEverything() {
132.     // while we have anything, drop all of what the first thing is.
133.     while (!isEmpty()) {
134.         leaveAll(whatIs(0));
135.     }
136. }
137. /** Return the kind of thing at particular point in the linkedList.
138.  * @param pos zero-based index.
139.  * @return kind of thing at that position.
140.  * @throws illegalStateException if position is out of range.
141.  */
142. public ThingType whatIs(int pos) {
143.     if (pos < 0) {
144.         throw new IllegalStateException("list what_is position negative");
145.     }
146.     CollectorNode cur = head;
147.     for (int i = 0; i < pos; i++) {
148.         if (cur == null) {
149.             throw new IllegalStateException("list what_is position too large");
150.         }
151.         else {
152.             cur = cur.next;
153.         }
154.     }
155.     return cur.thing;
156. }
157. /** Return the position of this kind of thing in the linkedList.
158.  * @param loc_thing kind of thing to look for.
159.  * @return zero-based index.
160.  * @throws illegalStateException if nothing of that type is present.
161.  */
162. public int whereIs(ThingType loc_thing) {
163.     int i = 0;
164.     CollectorNode cur = head;
165.     while (cur != null && loc_thing != cur.thing) {
166.         cur = cur.next;
167.         i++;
168.     }
169.     if (cur == null) {
170.         throw new IllegalStateException("Thing not found");
171.     }
172.     else return i;
173. }
174. /** Provide a explanation of what is in the collection.
175.  * @return human-readable string.
176.  */
177. public String show() {
178.     String s = "";
179.     if (!this.isEmpty()) {
180.         s += "Collector has ";
181.         int how_many = howManyThings();
182.         for (int j = 0; j < how_many; j++) {
183.             ThingType t = this.whatIs(j);
184.             int how_many_of_this = this.howManyOf(t);
185.             s += how_many_of_this + " " + ThingType.describe(t);
186.             if (how_many_of_this > 1) {
187.                 s += "s";
188.             }
189.             if (j < how_many - 1) {
190.                 s += ", ";
191.             }
192.             if (j == (how_many - 2)) {

```

```
193. s += "and ";
194. }
195. }
196. }
197. else {
198. s = "Collector has nothing.";
199. }
200. return s;
201. }
202. }
203.
```