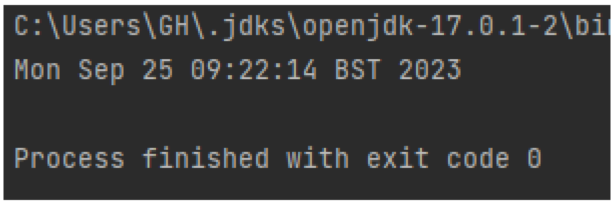
Exercise 1: Hello with Date Time

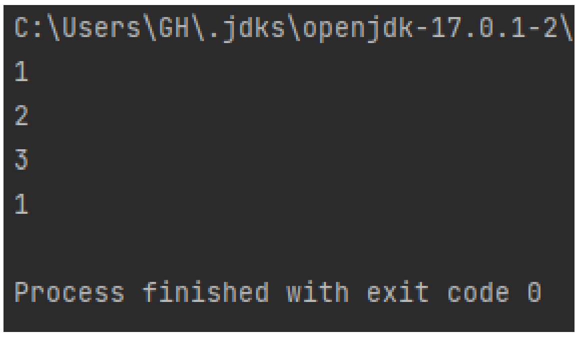


```
C:\Users\GH\.jdks\openjdk-17.0.1-2\bi
Mon Sep 25 09:22:14 BST 2023

Process finished with exit code 0
```

I expect that the values would be:

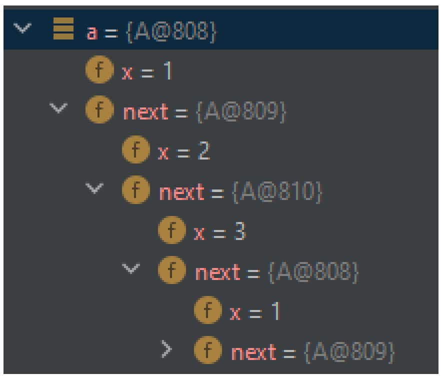
- 1
- 2
- 3
- 1

Result:



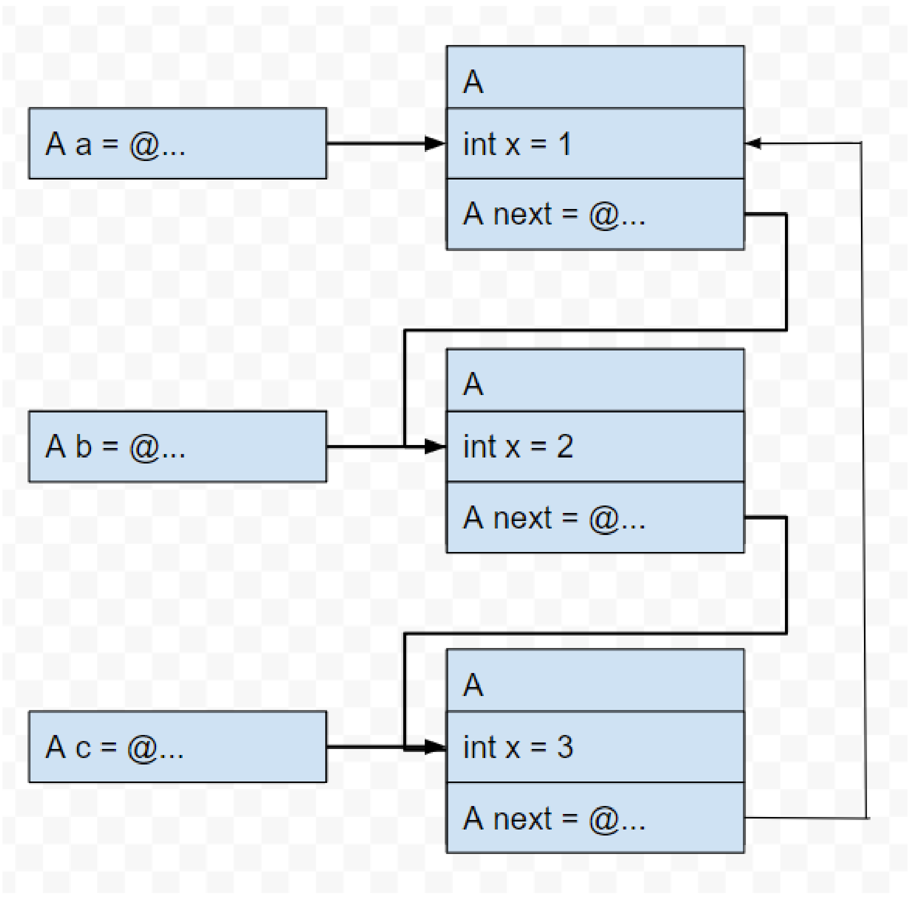
```
C:\Users\GH\.jdks\openjdk-17.0.1-2\
1
2
3
1

Process finished with exit code 0
```

In debugger, when expanding the variable a, it will loop values in variable a,b,c as they reference each other.



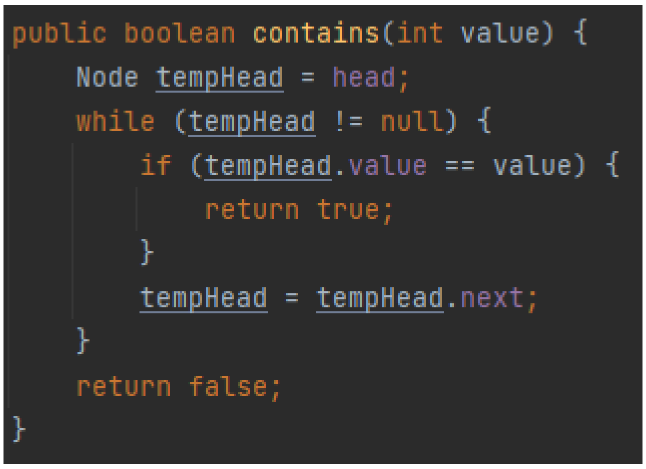
```
a = {A@808}
    x = 1
    next = {A@809}
        x = 2
        next = {A@810}
            x = 3
            next = {A@808}
                x = 1
                next = {A@809}
```

Exercise 2: Sketching Object Graphs



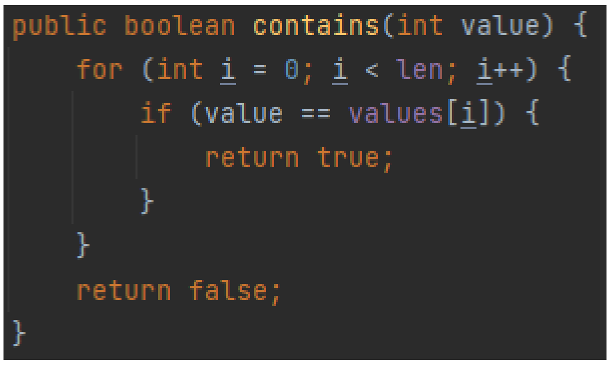
Exercise 3: Complete the Inefficient Implementations of `LinkedList` and `ArrayList`.

Contains method for LinkedList



```
public boolean contains(int value) {
    Node tempHead = head;
    while (tempHead != null) {
        if (tempHead.value == value) {
            return true;
        }
        tempHead = tempHead.next;
    }
    return false;
}
```

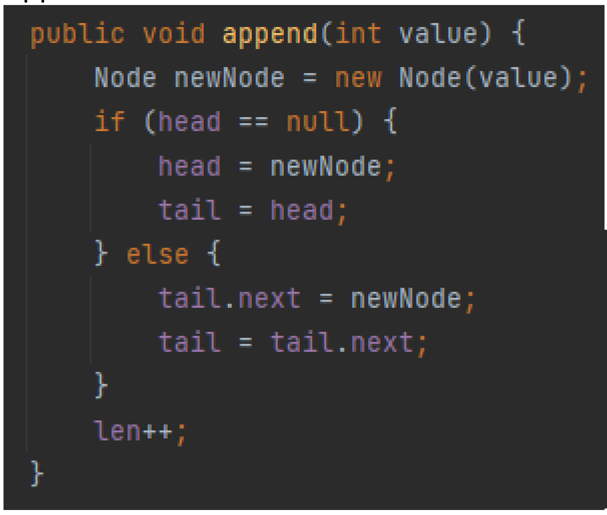
Contains method for ArrayList



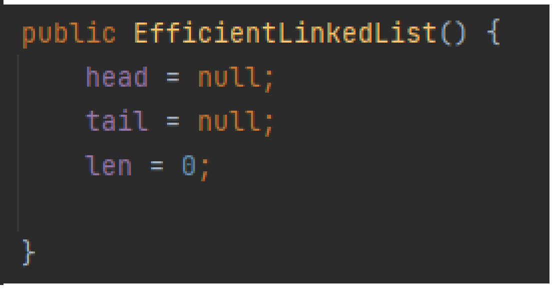
```java
public boolean contains(int value) {
    for (int i = 0; i < len; i++) {
        if (value == values[i]) {
            return true;
        }
    }
    return false;
}
```

Exercise 4: Minimal rewrite of `LinkedList` and `ArrayList` to improve efficiency

Append function for EfficientLinkedList

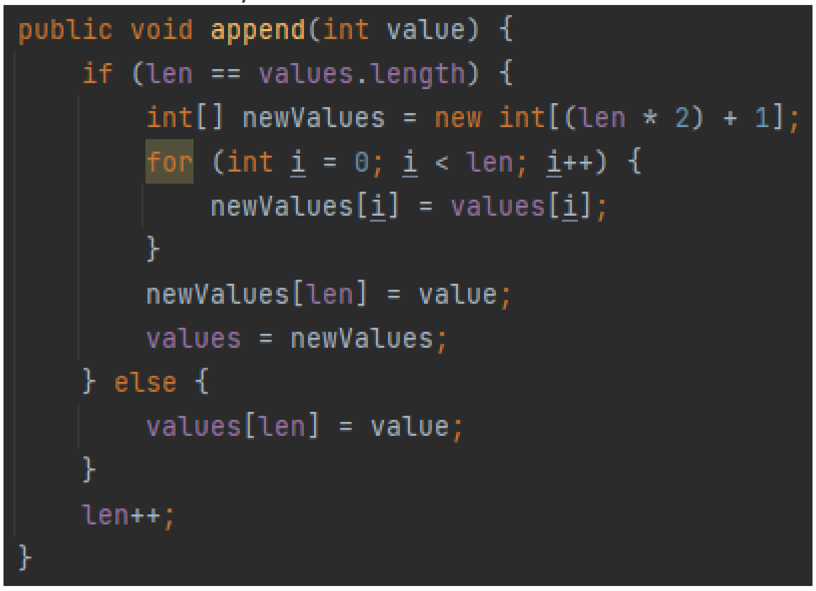


```java
public void append(int value) {
    Node newNode = new Node(value);
    if (head == null) {
        head = newNode;
        tail = head;
    } else {
        tail.next = newNode;
        tail = tail.next;
    }
    len++;
}
```



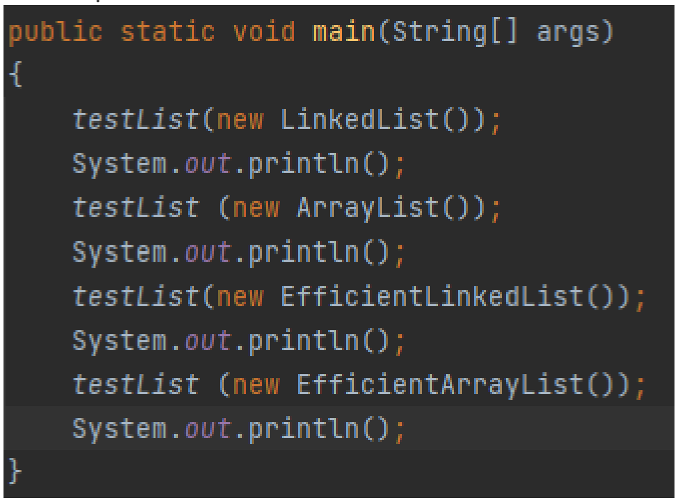
```java
public EfficientLinkedList() {
    head = null;
    tail = null;
    len = 0;

}
```

Append function for EfficientArrayList



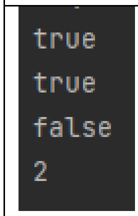
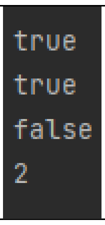
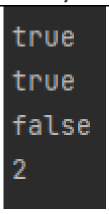
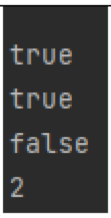
```java
public void append(int value) {
    if (len == values.length) {
        int[] newValues = new int[(len * 2) + 1];
        for (int i = 0; i < len; i++) {
            newValues[i] = values[i];
        }
        newValues[len] = value;
        values = newValues;
    } else {
        values[len] = value;
    }
    len++;
}
```

Code extension of ListExample



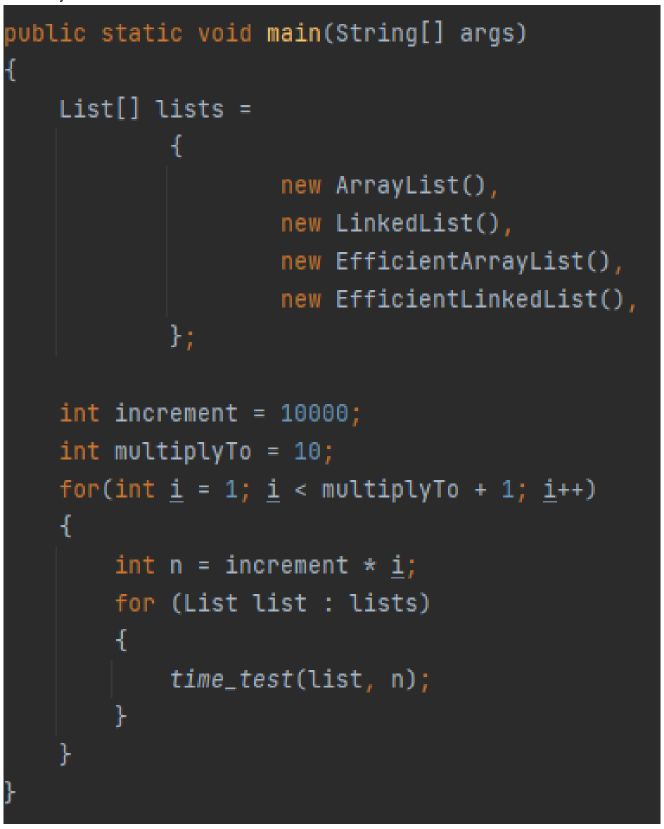
```java
public static void main(String[] args)
{
    testList(new LinkedList());
    System.out.println();
    testList (new ArrayList());
    System.out.println();
    testList(new EfficientLinkedList());
    System.out.println();
    testList (new EfficientArrayList());
    System.out.println();
}
```

Output using an extension of ListExample

LinkedList	EfficientLinkedList	ArrayList	EfficientArrayList
 true true false 2	 true true false 2	 true true false 2	 true true false 2

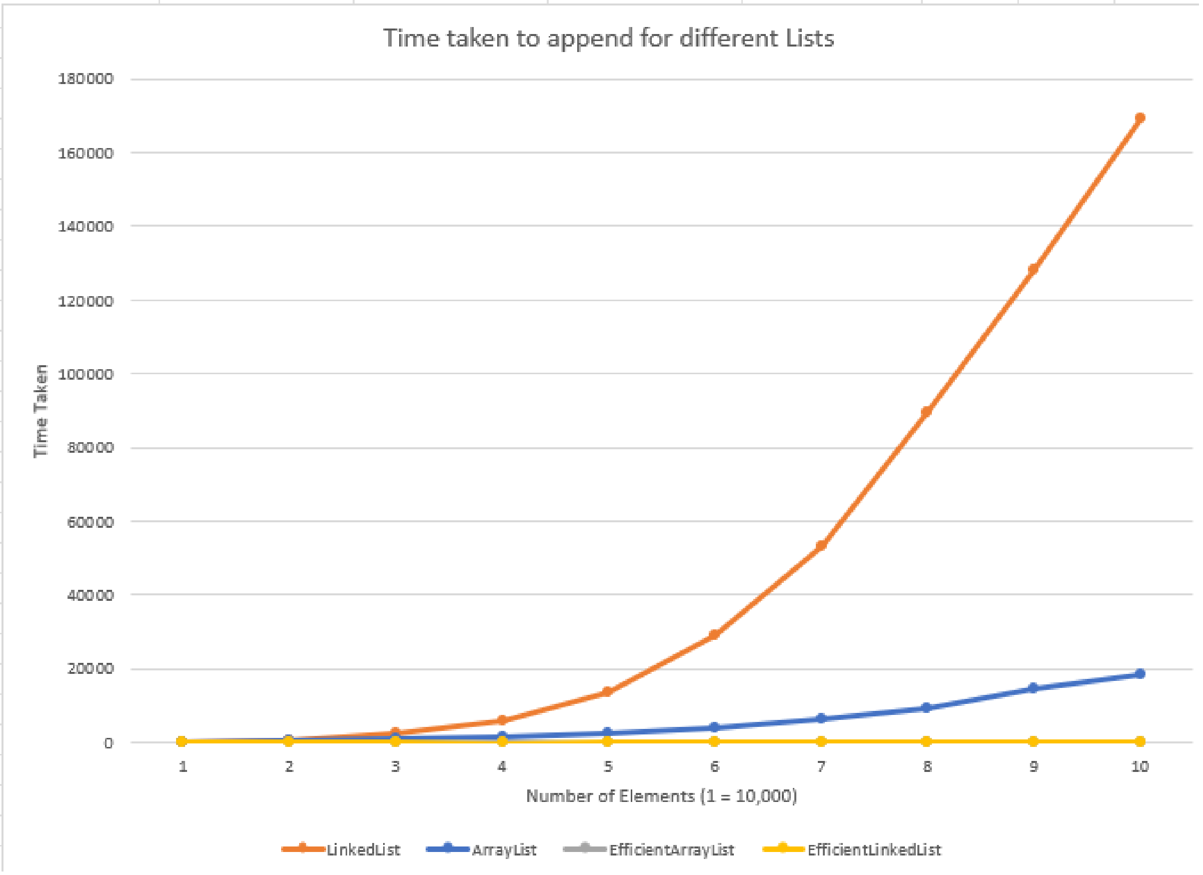
Exercise 5: Performance Tests

Code used to test (slightly modified so I don't have to change the code multiple times for a new number of elements)



```java
public static void main(String[] args)
{
    List[] lists =
            {
                    new ArrayList(),
                    new LinkedList(),
                    new EfficientArrayList(),
                    new EfficientLinkedList(),
            };

    int increment = 10000;
    int multiplyTo = 10;
    for(int i = 1; i < multiplyTo + 1; i++)
    {
        int n = increment * i;
        for (List list : lists)
        {
            time_test(list, n);
        }
    }
}
```

Elements	ArrayList	LinkedList	EfficientArrayList	EfficientLinkedList
10000	73	74	1	1
20000	270	626	0	1
30000	783	2173	0	1
40000	1258	5628	1	0
50000	2266	13755	0	1
60000	3831	29086	0	2
70000	6206	53300	1	1
80000	9213	89295	0	1
90000	14702	128081	0	1
100000	18491	169420	2	1



The average of EfficienctArrayList is constant while EfficientLinkedList for any case of append is constant