Example Problems: Error Messages

1. You create a program to convert the numbers [3, 24, 70, 50, 19] to letters using the LetterSorter. When you run the program you get an error that "70 cannot be converted to a letter".

```
Exception in thread "main" java.lang.RuntimeException: Invalid number: 70 cannot be converted to a letter.
        at lettersort.LetterSorter.convertNumbersToLetters(LetterSorter.java:43)
        at Main.main(Main.java:17)
```

LetterSorter.java:

```
1 package lettersort;
2
3 import java.util.ArrayList;
4 import java.util.Arrays;
5 import java.util.List;
6
7 public class LetterSorter {
8     public final List<String> CHAR_LIST = Arrays.asList(
9         "a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m",
10        "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z",
11        "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M",
12        "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z");
13
14    public List<Integer> convertLettersToNumbers(List<String> letterList) {
15      List<Integer> numberList = new ArrayList<Integer>();
16      // Loop through the letters and convert it to a number.
17      for (int index = 0; index < letterList.size(); index++) {
18        for (int charIndex = 0; charIndex < CHAR_LIST.size(); charIndex++) {
19          // If the character in the CHAR_LIST matches add the index to the numberList.
20          if (CHAR_LIST.get(charIndex) == letterList.get(index)) {
21              numberList.add(charIndex);
22              continue;
23          }
24        }
25
26        // The character was not found so throw an error.
27        if (numberList.size() != index + 1) {
28            throw new RuntimeException(
29                String.format("Invalid character: %s cannot be converted to a number.",
30                              letterList.get(index)));
```

```
31          }
32        }
33
34        return numberList;
35      }
36
37      public List<String> convertNumbersToLetters(List<Integer> numberList) {
38                List<String> letterList = new ArrayList<String>();
39        for (int index = 0; index < numberList.size(); index++) {
40            // The number does not match a character so throw an error.
41            if (numberList.get(index) > CHAR_LIST.size() || numberList.get(index) < 0) {
42                throw new RuntimeException(
43                    String.format("Invalid number: %d cannot be converted to a letter.",
44                                  numberList.get(index)));
45            }
46            // Look up the character that matches the number in the CHAR_LIST.
47            letterList.add(CHAR_LIST.get(numberList.get(index)));
48        }
49        return letterList;
50      }
51  }
```

2. Before you can run the MergeSorter program with the input [1, 4, 6, 2, 3], you see the error message “int cannot be converted to List<Integer>”.

```
Main.java:19: error: incompatible types: int cannot be converted to List<Integer>
     List<Integer> leftSorted = mergeSort(midpoint);
                                          ^
1 error
```

MergeSorter.java:

```
1 import java.util.ArrayList;
2 import java.util.Arrays;
3 import java.util.List;
4
5 class MergeSorter {
6
7   List<Integer> mergeSort(List<Integer> inputArray) {
8       // There's only 1 element so return the array.
9       if (inputArray.size() < 2){
10            return inputArray;
11       }
```

```
12
13      // Find the midpoint and split into two lists.
14      int midpoint = inputArray.size() / 2;
15      List<Integer> left = inputArray.subList(0, midpoint);
16      List<Integer> right = inputArray.subList(midpoint, inputArray.size());
17
18      // Sort both lists.
19      List<Integer> leftSorted = mergeSort(midpoint);
20      List<Integer> rightSorted = mergeSort(right);
21
22      // Merge the sorted lists by keeping a pointer to the first unused number
in each list.
23      // Compare the first unused item in each list and add the lowest to the
combined list.
24      // Stop when the end of either list has been reached.
25      ArrayList<Integer> combinedSorted = new ArrayList<Integer>();
26      int leftIndex = 0;
27      int rightIndex = 0;
28      while (leftIndex < leftSorted.size() && rightIndex < rightSorted.size()) {
29          if (leftSorted.get(leftIndex) <= rightSorted.get(rightIndex)){
30              combinedSorted.add(leftSorted.get(leftIndex));
31              leftIndex++;
32          } else {
33              combinedSorted.add(rightSorted.get(rightIndex));
34              rightIndex++;
35          }
36      }
37
38      // Add the remaining items in the list where the end was not reached.
39      if (leftIndex == leftSorted.size()){
40          combinedSorted.addAll(rightSorted.subList(rightIndex, right.size()));
41      } else {
42          combinedSorted.addAll(leftSorted.subList(leftIndex, left.size()));
43      }
44
45      return combinedSorted;
46    }
47  }
```

3. N/A (These are the same problems used in the previous small group activity. N/A is used to signal an error that does not apply to this small group activity while preserving the numbering across all small group activities.)

4. When you try to run the MergeSorter program with the list [1, 4, 6, 2, 3], an error message pops up saying “cannot find symbol” before the program runs.

```
Main.java:19: error: cannot find symbol
     List<Integer> leftSorted = mergesort(left);
                                ^
  symbol:   method mergesort(List<Integer>)
  location: class MergeSorter
1 error
```

MergeSorter.java:

```
1 import java.util.ArrayList;
2 import java.util.Arrays;
3 import java.util.List;
4
5 class MergeSorter {
6
7  List<Integer> mergeSort(List<Integer> inputArray) {
8     // There's only 1 element so return the array.
9     if (inputArray.size() < 2){
10         return inputArray;
11     }
12
13     // Find the midpoint and split into two lists.
14     int midpoint = inputArray.size() / 2;
15     List<Integer> left = inputArray.subList(0, midpoint);
16     List<Integer> right = inputArray.subList(midpoint, inputArray.size());
17
18     // Sort both lists.
19     List<Integer> leftSorted = mergesort(left);
20     List<Integer> rightSorted = mergeSort(right);
21
22     // Merge the sorted lists by keeping a pointer to the first unused number in
each list.
23     // Compare the first unused item in each list and add the lowest to the
combined list.
24     // Stop when the end of either list has been reached.
25    ArrayList<Integer> combinedSorted = new ArrayList<Integer>();
26     int leftIndex = 0;
27     int rightIndex = 0;
28     while (leftIndex < leftSorted.size() && rightIndex < rightSorted.size()) {
29         if (leftSorted.get(leftIndex) <= rightSorted.get(rightIndex)){
30             combinedSorted.add(leftSorted.get(leftIndex));
```

```
            leftIndex++;
        } else {
            combinedSorted.add(rightSorted.get(rightIndex));
            rightIndex++;
        }
    }

    // Add the remaining items in the list where the end was not reached.
    if (leftIndex == leftSorted.size()){
        combinedSorted.addAll(rightSorted.subList(rightIndex, right.size()));
    } else {
        combinedSorted.addAll(leftSorted.subList(leftIndex, left.size()));
    }

    return combinedSorted;
 }
}

```

5. You go to build the MergeSorter program and immediately see an error message " ')' expected".

```
Main.java:39: error: ')' expected
     if (leftIndex == leftSorted.size(){
                                      ^
1 error
```

MergeSorter.java:

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

class MergeSorter {

  List<Integer> mergeSort(List<Integer> inputArray) {
     // There's only 1 element so return the array.
     if (inputArray.size() < 2){
          return inputArray;
```

```
11      }
12
13      // Find the midpoint and split into two lists.
14      int midpoint = inputArray.size() / 2;
15      List<Integer> left = inputArray.subList(0, midpoint);
16      List<Integer> right = inputArray.subList(midpoint, inputArray.size());
17
18      // Sort both lists.
19      List<Integer> leftSorted = mergeSort(left);
20      List<Integer> rightSorted = mergeSort(right);
21
22      // Merge the sorted lists by keeping a pointer to the first unused number
in each list.
23      // Compare the first unused item in each list and add the lowest to the
combined list.
24      // Stop when the end of either list has been reached.
25     ArrayList<Integer> combinedSorted = new ArrayList<Integer>();
26      int leftIndex = 0;
27      int rightIndex = 0;
28      while (leftIndex < leftSorted.size() && rightIndex < rightSorted.size()) {
29          if (leftSorted.get(leftIndex) <= rightSorted.get(rightIndex)){
30              combinedSorted.add(leftSorted.get(leftIndex));
31              leftIndex++;
32          } else {
33              combinedSorted.add(rightSorted.get(rightIndex));
34              rightIndex++;
35          }
36      }
37
38      // Add the remaining items in the list where the end was not reached.
39      if (leftIndex == leftSorted.size(){
40          combinedSorted.addAll(rightSorted.subList(rightIndex, right.size()));
41      } else {
42          combinedSorted.addAll(leftSorted.subList(leftIndex, left.size()));
43      }
44
45      return combinedSorted;
46    }
47 }
```

6. N/A
7. When you try to build the MergeSorter program you get an error message saying "incompatible types: int cannot be converted to boolean".

```
Main.java:39: error: incompatible types: int cannot be converted to boolean
     if (leftIndex = leftSorted.size()){
                   ^
1 error
```

MergeSorter.java:

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

class MergeSorter {

  List<Integer> mergeSort(List<Integer> inputArray) {
     // There's only 1 element so return the array.
     if (inputArray.size() < 2){
          return inputArray;
      }

     // Find the midpoint and split into two lists.
     int midpoint = inputArray.size() / 2;
     List<Integer> left = inputArray.subList(0, midpoint);
     List<Integer> right = inputArray.subList(midpoint, inputArray.size() + 1);

     // Sort both lists.
     List<Integer> leftSorted = mergeSort(left);
     List<Integer> rightSorted = mergeSort(right);

     // Merge the sorted lists by keeping a pointer to the first unused number in each list.
     // Compare the first unused item in each list and add the lowest to the combined list.
     // Stop when the end of either list has been reached.
     ArrayList<Integer> combinedSorted = new ArrayList<Integer>();
     int leftIndex = 0;
     int rightIndex = 0;
     while (leftIndex < leftSorted.size() && rightIndex < rightSorted.size()) {
         if (leftSorted.get(leftIndex) <= rightSorted.get(rightIndex)){
             combinedSorted.add(leftSorted.get(leftIndex));
             leftIndex++;
         } else {
             combinedSorted.add(rightSorted.get(rightIndex));
             rightIndex++;
         }
     }

     // Add the remaining items in the list where the end was not reached.
```

```
39      if (leftIndex = leftSorted.size()){
40          combinedSorted.addAll(rightSorted.subList(rightIndex, right.size()));
41      } else {
42          combinedSorted.addAll(leftSorted.subList(leftIndex, left.size()));
43      }
44
45      return combinedSorted;
46    }
47 }
```

8. When you try to run the LetterSorter program with the list ["c", "b", "e", "A", "M", "i", "?"] you get an error message while running that there is an “invalid character”.

```
Exception in thread "main" java.lang.RuntimeException: Invalid character: ? cannot
be converted to a number.
        at lettersort.LetterSorter.convertLettersToNumbers(LetterSorter.java:29)
        at Main.main(Main.java:15)
```

LetterSorter.java:

```
1 package lettersort;
2
3 import java.util.ArrayList;
4 import java.util.Arrays;
5 import java.util.List;
6
7 public class LetterSorter {
8     public final List<String> CHAR_LIST = Arrays.asList(
9         "a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m",
10        "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z",
11        "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M",
12        "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z");
13
14     public List<Integer> convertLettersToNumbers(List<String> letterList) {
15       List<Integer> numberList = new ArrayList<Integer>();
16       // Loop through the letters and convert it to a number.
17       for (int index = 0; index < letterList.size(); index++) {
18         for (int charIndex = 0; charIndex < CHAR_LIST.size(); charIndex++) {
19           // If the character in the CHAR_LIST matches add the index to the numberList.
20           if (CHAR_LIST.get(charIndex) == letterList.get(index)) {
21               numberList.add(charIndex);
22               continue;
23           }
```

```
24          }
25
26          // The character was not found so throw an error.
27          if (numberList.size() != index + 1) {
28              throw new RuntimeException(
29                  String.format("Invalid character: %s cannot be converted to a number.",
30                                letterList.get(index)));
31          }
32        }
33
34        return numberList;
35      }
36
37      public List<String> convertNumbersToLetters(List<Integer> numberList) {
38                List<String> letterList = new ArrayList<String>();
39        for (int index = 0; index < numberList.size(); index++) {
40            // The number does not match a character so throw an error.
41            if (numberList.get(index) > CHAR_LIST.size() || numberList.get(index) < 0) {
42                throw new RuntimeException(
43                    String.format("Invalid number: %d cannot be converted to a letter.",
44                                  numberList.get(index)));
45            }
46            // Look up the character that matches the number in the CHAR_LIST.
47            letterList.add(CHAR_LIST.get(numberList.get(index)));
48        }
49        return letterList;
50      }
51  }
```

9. You're able to build the MergeSorter program but when you go to run it you get an error that the index is out of bounds.

```
Exception in thread "main" java.lang.IndexOutOfBoundsException: toIndex = 2
        at java.base/java.util.AbstractList.subListRangeCheck(AbstractList.java:507)
        at java.base/java.util.AbstractList$RandomAccessSubList.subList(AbstractList.java:938)
        at mergesort.MergeSorter.mergeSort(MergeSorter.java:42)
        at mergesort.MergeSorter.mergeSort(MergeSorter.java:21)
        at Main.main(Main.java:26)
```

MergeSorter.java

```
1 package mergesort;
```

```java

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class MergeSorter {

  public List<Integer> mergeSort(List<Integer> inputArray) {
      // There's only 1 element so return the array.
      if (inputArray.size() < 2){
          return inputArray;
      }

      // Find the midpoint and split into two lists.
      int midpoint = inputArray.size() / 2;
      List<Integer> left = inputArray.subList(0, midpoint);
      List<Integer> right = inputArray.subList(midpoint, inputArray.size());

      // Sort both lists.
      List<Integer> leftSorted = mergeSort(left);
      List<Integer> rightSorted = mergeSort(right);

      // Merge the sorted lists by keeping a pointer to the first unused number in each list.
      // Compare the first unused item in each list and add the lowest to the combined list.
      // Stop when the end of either list has been reached.
      ArrayList<Integer> combinedSorted = new ArrayList<Integer>();
      int leftIndex = 0;
      int rightIndex = 0;
      while (leftIndex < leftSorted.size() && rightIndex < rightSorted.size()) {
          if (leftSorted.get(leftIndex) <= rightSorted.get(rightIndex)){
              combinedSorted.add(leftSorted.get(leftIndex));
              leftIndex++;
          } else {
              combinedSorted.add(rightSorted.get(rightIndex));
              rightIndex++;
          }
      }

      // Add the remaining items in the list where the end was not reached.
      if (leftIndex == leftSorted.size()) {
          combinedSorted.addAll(rightSorted.subList(rightIndex, rightSorted.size() + 1));
      } else {
          combinedSorted.addAll(leftSorted.subList(leftIndex, leftSorted.size()));
      }

      return combinedSorted;
  }
```

```
49 }
```