Solution 1

```java
// Copyright © 2020 AlgoExpert, LLC. All rights reserved.

import java.util.*;

class Program {
  // O(b + s) time | O(b + s) space - where b is the length of the big
  // input string and s is the length of the small input string
  public static String smallestSubstringContaining(String bigString, String smallString) {
    Map<Character, Integer> targetCharCounts = getCharCounts(smallString);
    List<Integer> substringBounds = getSubstringBounds(bigString, targetCharCounts);
    return getStringFromBounds(bigString, substringBounds);
  }

  public static Map<Character, Integer> getCharCounts(String string) {
    Map<Character, Integer> charCounts = new HashMap<Character, Integer>();
    for (int i = 0; i < string.length(); i++) {
      increaseCharCount(string.charAt(i), charCounts);
    }
    return charCounts;
  }

  public static List<Integer> getSubstringBounds(
      String string, Map<Character, Integer> targetCharCounts) {
    List<Integer> substringBounds = new ArrayList<Integer>(Arrays.asList(0, Integer.MAX_VALUE));
    Map<Character, Integer> substringCharCounts = new HashMap<Character, Integer>();
    int numUniqueChars = targetCharCounts.size();
    int numUniqueCharsDone = 0;
    int leftIdx = 0;
    int rightIdx = 0;
    // Move the rightIdx to the right in the string until you've counted
    // all of the target characters enough times.
    while (rightIdx < string.length()) {
      char rightChar = string.charAt(rightIdx);
      if (!targetCharCounts.containsKey(rightChar)) {
        rightIdx++;
        continue;
      }
      increaseCharCount(rightChar, substringCharCounts);
      if (substringCharCounts.get(rightChar) == targetCharCounts.get(rightChar)) {
        numUniqueCharsDone++;
      }
      // Move the leftIdx to the right in the string until you no longer
      // have enough of the target characters in between the leftIdx and
      // the rightIdx. Update the substringBounds accordingly.
      while (numUniqueCharsDone == numUniqueChars && leftIdx <= rightIdx) {
        substringBounds =
            getCloserBounds(leftIdx, rightIdx, substringBounds.get(0), substringBounds.get(1));
        char leftChar = string.charAt(leftIdx);
        if (!targetCharCounts.containsKey(leftChar)) {
          leftIdx++;
          continue;
        }
        if (substringCharCounts.get(leftChar) == targetCharCounts.get(leftChar)) {
          numUniqueCharsDone--;
        }
        decreaseCharCount(leftChar, substringCharCounts);
        leftIdx++;
      }
      rightIdx++;
    }
    return substringBounds;
  }

  public static List<Integer> getCloserBounds(int idx1, int idx2, int idx3, int idx4) {
    return idx2 - idx1 < idx4 - idx3
        ? new ArrayList<Integer>(Arrays.asList(idx1, idx2))
        : new ArrayList<Integer>(Arrays.asList(idx3, idx4));
  }

  public static String getStringFromBounds(String string, List<Integer> bounds) {
    int start = bounds.get(0);
    int end = bounds.get(1);
    if (end == Integer.MAX_VALUE) return "";
    return string.substring(start, end + 1);
  }

  public static void increaseCharCount(char c, Map<Character, Integer> charCounts) {
    if (!charCounts.containsKey(c)) {
      charCounts.put(c, 1);
    } else {
      charCounts.put(c, charCounts.get(c) + 1);
    }
  }

  public static void decreaseCharCount(char c, Map<Character, Integer> charCounts) {
    charCounts.put(c, charCounts.get(c) - 1);
  }
}
```