Java Code Solutions to Interview Problems

1. 1. Longest substring without repeating characters

```
public int lengthOfLongestSubstring(String s) {     int[] index = new int[128];
   int maxLen = 0,
start = 0;     for (int end = 0; end < s.length(); end++) {         start =
Math.max(index[s.charAt(end)], start);           maxLen = Math.max(maxLen, end -
start + 1);
index[s.charAt(end)] = end + 1;     }     return maxLen; }
```

2. 2. Longest palindromic substring

```
public String longestPalindrome(String s) {     int start = 0, end = 0;     for
(int i = 0; i <
s.length(); i++) {          int len1 = expandAroundCenter(s, i, i);          int
len2 =
expandAroundCenter(s, i, i + 1);         int len = Math.max(len1, len2);
if (len > end -
start) {             start = i - (len - 1) / 2;             end = i + len / 2;
      }     }
return s.substring(start, end + 1); } private int expandAroundCenter(String s,
int left, int right)
{          while (left >= 0 && right < s.length() && s.charAt(left) ==
s.charAt(right)) {         left--;
right++;     }     return right - left - 1; }
```

3. 3. Check if two strings are anagrams

```
public boolean isAnagram(String s, String t) {     if (s.length() != t.length())
return false;
int[] count = new int[26];     for (int i = 0; i < s.length(); i++) {
count[s.charAt(i) -
'a']++;         count[t.charAt(i) - 'a']--;     }     for (int c : count) {
   if (c != 0)
return false;     }     return true; }
```

4. 4. Valid palindrome check

```
public boolean isPalindrome(String s) {     int left = 0, right = s.length() -
1;     while (left <
right)    {                                  while    (left    <    right    &&
!Character.isLetterOrDigit(s.charAt(left))) left++;
while (left < right && !Character.isLetterOrDigit(s.charAt(right))) right--;
    if
(Character.toLowerCase(s.charAt(left))                                        !=
Character.toLowerCase(s.charAt(right)))             return
false;         left++;         right--;     }     return true; }
```

5. 5. String compression (e.g., aabcccccaaa -> a2b1c5a3)

```
public String compress(String s) {     StringBuilder sb = new StringBuilder();
```

```
  int count = 1;
for (int i = 1; i <= s.length(); i++) {             if (i < s.length() &&
s.charAt(i) == s.charAt(i -
1)) {              count++;          } else {              sb.append(s.charAt(i -
1)).append(count);
count = 1;          }     }     return sb.toString(); }
```

6. 6. Count and say sequence

```
public String countAndSay(int n) {     String res = "1";     for (int i = 1; i <
n; i++) {
StringBuilder sb = new StringBuilder();          int count = 1;          for (int
j = 1; j <=
res.length(); j++) {                  if (j < res.length() && res.charAt(j) ==
res.charAt(j - 1)) {
count++;                              }  else  {
sb.append(count).append(res.charAt(j - 1));
count = 1;             }         }         res = sb.toString();     }     return
res; }
```

7. 7. Implement string to integer (atoi)

```
public int myAtoi(String s) {     int i = 0, sign = 1, result = 0;     while (i
< s.length() &&
s.charAt(i) == ' ') i++;       if (i < s.length() && (s.charAt(i) == '+' ||
s.charAt(i) == '-'))
sign = s.charAt(i++) == '-' ? -1 : 1;          while (i < s.length() &&
Character.isDigit(s.charAt(i))) {
int digit = s.charAt(i++) - '0';            if (result > (Integer.MAX_VALUE -
digit) / 10)
return sign == 1 ? Integer.MAX_VALUE : Integer.MIN_VALUE;            result =
result * 10 + digit;
}     return result * sign; }
```

8. 8. Implement strstr() / indexOf()

```
public int strStr(String haystack, String needle) {      if (needle.isEmpty())
return 0;     for (int
i = 0; i <= haystack.length() - needle.length(); i++) {                if
(haystack.substring(i, i +
needle.length()).equals(needle)) return i;     }     return -1; }
```

9. 9. Group anagrams

```
public List<List<String>> groupAnagrams(String[] strs) {          Map<String,
List<String>> map = new
HashMap<>();     for (String s : strs) {          char[] arr = s.toCharArray();
Arrays.sort(arr);                String key = new String(arr);
map.computeIfAbsent(key, k -> new
ArrayList<>()).add(s);     }     return new ArrayList<>(map.values()); }
```

10. 10. Reverse words in a string

```
public String reverseWords(String s) { String[] words = s.trim().split("\s+");
Collections.reverse(Arrays.asList(words)); return String.join(" ", words); }
```

11. 11. Check if two strings are isomorphic

```
public boolean isIsomorphic(String s, String t) { if (s.length() != t.length()) return false;
Map<Character, Character> map = new HashMap<>(); Set<Character> used = new HashSet<>(); for
(int i = 0; i < s.length(); i++) { char c1 = s.charAt(i), c2 = t.charAt(i); if
(map.containsKey(c1)) { if (map.get(c1) != c2) return false;
} else {
if (used.contains(c2)) return false; map.put(c1, c2);
used.add(c2);
} } return true; }
```

12. 12. Longest common prefix

```
public String longestCommonPrefix(String[] strs) { if (strs == null || strs.length == 0) return
""; String prefix = strs[0]; for (int i = 1; i < strs.length; i++) {
while
(strs[i].indexOf(prefix) != 0) { prefix = prefix.substring(0, prefix.length() - 1);
if (prefix.isEmpty()) return ""; } } return prefix; }
```