1. Explain the goals of the API peer review method. How is the method different from API usability testing?

The goal of an API peer review is to get feedback on the usability of an Application programming interface, API, (or some part of “feature” of the API). The usability on an API can be defined in terms of several aspects of using the API by developers (e.g. the cognitive dimensions framework) to build a solution to a specific problem that the API is supposed to help to solve (such as using networking sockets in a program). A peer review provides an expert assessment of the usability of an API and it is also meant to uncover usability bugs.

API usability testing is performed in laboratory environment, and the idea is that real developers are given a task to implement real code using the API under test. The working of the developer is monitored during the test to uncover any problems the developer has in using the API.

API Peer review does not require a lab environment, or a long lead/preparation time before the test, and nor does it require a lengthy analysis of the observations/recordings after the test. Therefore, reviews require considerably less human resources, equipment, and time. Hence, a larger number of APIs can be covered with reviews than usability tests in a development project that is under time and budget pressure. The results of usability reviews are comparable to the results achieved with usability tests, but more problems related to the first exposure and initial learning curve of the API are found with usability reviews.

2. Describe the phases of the review process.

Roles:
- Feature owner – developer/designer of the API (or feature) under review
- Feature area manager – the superior of the former, API area owner
- Usability engineer – the usability expert who knows what API usability is and how it’s measured; leads the review process
- Reviewers – chosen based on their expertise in the API area

Process

1. Planning (usability engineer & feature owner)
Define the objectives, particular questions the feature owner wants feedback on
Get a code sample of the API/feature under review
Choose reviewers
Decide logistics

2. Review meeting (the whole review team)
• 1.5 hour meeting
• List the goals of the review to everybody
• Explain the background of the API
• The feature owner walks the reviewers through typical actions with the API, and the reviewers are prompted to consider any mismatches between what the API is supposed to do on a conceptual level from a developers point of view and how it is actually represented by the code they need to write while using the API
• Usability engineer guides the feedback of the reviewers towards usability problems
• Key questions in the review are 1) what do you think this code snippet does, and 2) does the implementation make sense?

3. Bug filing (usability engineer & feature owner & feature area manager)
• 1 hour meeting
• Issues found in the review meeting are gone through and appropriate bug reports opened and prioritized by the feature area manager

3. Metodi splitInTwo() jakaa syötteen annetun merkkijonon original kahteen osaan merkin marker ensimmäisen ilmentymän kohdalta ja palauttaa osat taulukossa. Merkin marker ilmentymä on poistettu katkaisukohdasta.

The method splitInTwo() divides the input string original into two parts at the first appearance of the character marker and returns the parts in an array. The marker character is removed from the position where the input string is split.

```java
public static String[] splitInTwo(String original, char marker) {
    int i = 0, pos = 0; char ch = ' '; boolean found = false;
    for (; i < original.length(); i++) {
        ch = original.charAt(i);
        if (ch == marker) {
            pos = i;
            break;
        }
    }
    String[] result = new String[2];
    result[0] = original.substring(0, pos);
    result[1] = original.substring(pos + 1);
    return result;
}
```

Tee metodin muuttujille datavuoanalyysi. Mitä anomalioita löydät? Näyttäisikö niihin liittyvän vikoja?
Perform a data flow analysis for the variables used in the method. Can you find any anomalies? Do they indicate actual faults in the code?

Anomalies:

- **DU anomaly** for variable found. This clearly looks like a bug, the developer probably thought that he should check if the marker was actually found in the input string but he has forgot to do so. What actually is the relationship between the values of variables found and pos?
- **DD anomaly** for variable ch. Its initial value is not used before being overwritten. Looks OK, not a bug. The initial value is not needed, however, because ch is always assigned a value before being used (referenced) in the body of the **for** -loop.
- **DD anomaly** for variable pos, if marker is found in the string. Looks OK – BUT: an interesting question is, what is the value of pos in the case that marker is not found in the string and how that value is then used?
- **DU anomaly** for marker, if the input string has length 0. OK, not a bug.


**Apply Boundary Value thinking to the planning of test cases for the method** **splitInTwo()** **above. What kind of extreme inputs or border cases would you use to test this method? Consider also the interplay of both of the input parameters and think how it affects the choice of the test cases.**

**List of classes of inputs to consider**

**Boundary/Extreme values for input parameter original**

- NULL (original is an Object)
- empty string
- strings of length 1
- strings of length 2
- strings of length 3
- very long strings -> max. length is the maximum value of "int" 2147483647 (2^31), limit can be lower due to restrictions on the amount of memory available to the process executing the code in the runtime environment

**Boundary/Extreme values for input parameter marker**

- the minimum value of type char '\u0000' or 0
- one after the minimum value of type char '\u0001' or 1
- the maximum value of type char '\uffff' or 65535
- one before the maximum value of type char '\ufffe' or 65534

**original** and **marker** considered together, these should be combined with the cases above

- strings that contain at least one occurrence of the marker
  - strings that contain exactly one occurrence of the marker
  - strings that contain multiple occurrences of the marker
5. Määritä ekvivalenssiluokat ja raja-arvot seuraavan funktion syötteille:

"Funktio $\text{BMI}(\text{weight, height})$ laskee henkilön painoindeksin (BMI) kaavalla:

$$\text{weight} / (\text{height} \times \text{height})$$

Weight

$vEC1 : [1, ..., 800]$ $\text{BV} = \{0, 1, 2, 799, 800, 801\}$

$iEC1: [\text{MIN\_INT}, ..., 0] \text{ BV} = \{\text{MIN\_INT}-1, \text{MIN\_INT}, \text{MIN\_INT}+1, -1, 0, 1\}$

$iEC2: [801, ..., \text{MAX\_INT}] \text{ BV} = \{800, 801, 802, \text{MAX\_INT}-1, \text{MAX\_INT}, \text{MAX\_INT}+1\}$

$iEC3: \text{NaN}$

Height

$vEC2: [0, ..., 3]$ $\text{BV} = \{0, 0.01, 0.02, 2.99, 3.00, 3.01\}$

$iEC4: [\text{MIN\_DOUBLE}, ..., 0] \text{ BV} = \{\text{MIN\_DOUBLE}-0.01, \text{MIN\_DOUBLE}, \text{MIN\_DOUBLE}+0.01, -0.01, 0, 0.01\}$
iec5: 3, ..., MAX_DOUBLE

iec6: NaN