# User Study Task

You have 15 minutes to complete this task. The task is not to test how quickly you can finish the task. So please aim for the thoroughness, not speed. The experimenter will remind you when you have 5 minutes left.

**[5 min] Part I. Get familiar with the Stack Overflow post and its code example.**

Suppose you want to implement a function that computes the relative file path. You find [this SO post](https://stackoverflow.com/questions/204784/how-to-construct-a-relative-path-in-java-from-two-absolute-paths-or-urls#3054692) that illustrates how to implement the function in Java. Please inspect the code example in this post and focus on understanding its functionality and implementation details.

**[10 min] Part II. Create a reusable template of the SO code example.**

Now you want to create a template of the SO code example so other programmers do not have to repetitively adapt the code example to their own projects.

Please specify the following code parts in the given code example.

(1) Use the blue color to tag where you consider as the gist of the example and should remain unchanged during code reuse.

(2) Use the yellow color to tag where should be changed or removed based on the context of the target program.

(3) Use the red color to specify what additional code you would like to add into the template.

|  |
| --- |
| public static String getRelativePath(String targetPath, String basePath, String pathSeparator) {  // Normalize the paths  //null check for basepath  String normalizedTargetPath = FilenameUtils.normalizeNoEndSeparator(targetPath);  String normalizedBasePath = FilenameUtils.normalizeNoEndSeparator(basePath);  // Undo the changes to the separators made by normalization  if (pathSeparator.equals("/")) {  normalizedTargetPath = FilenameUtils.separatorsToUnix(normalizedTargetPath);  normalizedBasePath = FilenameUtils.separatorsToUnix(normalizedBasePath);  } else if (pathSeparator.equals("\\")) {  normalizedTargetPath =  FilenameUtils.separatorsToWindows(normalizedTargetPath);  normalizedBasePath = FilenameUtils.separatorsToWindows(normalizedBasePath);  } else {  throw new IllegalArgumentException("Unrecognised dir separator '" + pathSeparator + "'");  }  String[] base = normalizedBasePath.split(Pattern.quote(pathSeparator));  String[] target = normalizedTargetPath.split(Pattern.quote(pathSeparator));  // First get all the common elements. Store them as a string, // and also count how many of them there are.  StringBuffer common = new StringBuffer();  int commonIndex = 0;  while (commonIndex < target.length && commonIndex < base.length  && target[commonIndex].equals(base[commonIndex])) {  common.append(target[commonIndex] + pathSeparator);  commonIndex++;  }  if (commonIndex == 0) {  // No single common path element. This most  // likely indicates differing drive letters, like C: and D:.  // These paths cannot be relativized.  throw new PathResolutionException("No common path element found for '" + normalizedTargetPath + "' and '" + normalizedBasePath + "'");  }  // The number of directories we have to backtrack depends on whether the base is a file or a dir  // For example, the relative path from /foo/bar/baz/gg/ff to /foo/bar/baz  // ".." if ff is a file  // "../.." if ff is a directory  //  // The following is a heuristic to figure out if the base refers to a file or dir.  // It's not perfect, because the resource referred to by this path may not actually exist, but it's the best I can do  boolean baseIsFile = true;  File baseResource = new File(normalizedBasePath);  if (baseResource.exists()) {  baseIsFile = baseResource.isFile();  } else if (basePath.endsWith(pathSeparator)) {  baseIsFile = false;  }  StringBuffer relative = new StringBuffer();  if (base.length != commonIndex) {  int numDirsUp = baseIsFile ? base.length - commonIndex - 1 : base.length - commonIndex;  for (int i = 0; i < numDirsUp; i++) {  relative.append(".." + pathSeparator);  }  }  relative.append(normalizedTargetPath.substring(common.length()));  return relative.toString();  } |