```
/* Problem 3--Just Ken
   What I did here is find all the KENs and all the BARBIEs and checked
   for intersections once these were all found. */

import java.io.*;
import java.util.*;

public class prob3 {

 private static Scanner in;
 private static PrintWriter out;
 private static int cs;
 private static int rsz, csz;
 private static char[][] Grid;
 private static Tuple[] Kens;
 private static Tuple[] Barbies;
 private static int kct, bct;

 public static void main (String[] args) throws Exception {

  cs = 1;
  in = new Scanner (new File ("prob3.in"));
  out = new PrintWriter ("prob3.out");
  while (true) {
   rsz = in.nextInt();
   csz = in.nextInt();
   if (rsz==0 && csz==0) break;
   in.nextLine();
   Grid = new char[rsz][csz];
   for (int i=0; i < rsz; i++)
    Grid[i] = in.nextLine().toCharArray();
   Process ();
  }
  in.close ();
  out.close ();
 }

 public static void Process () throws Exception {

  out.printf ("Case %d:  ",cs++);
  int[] retsz = new int[1];
  Kens = GetString ("KEN",retsz);
  kct = retsz[0]; //get all KENS and BARBIES
  Barbies = GetString ("BARBIE",retsz);
  bct = retsz[0];
  int kwb = 0;
  for (int i=0; i < kct; i++) { //Go through all the KENs
   int drk = (Kens[i].getlastr()-Kens[i].getfirstr())/2;
   int dck = (Kens[i].getlastc()-Kens[i].getfirstc())/2;
   boolean touch = false;  //Get direction of KEN
   for (int j=0; j < bct && !touch; j++) {
    int drb = (Barbies[j].getlastr()-Barbies[j].getfirstr())/5;
    int dcb = (Barbies[j].getlastc()-Barbies[j].getfirstc())/5;
    for (int ii=0; ii < 3 && !touch; ii++) //Get dir of BARBIE
     for (int jj=0; jj < 6 && !touch; jj++) { //See if any letters touch
      touch = Math.abs (Kens[i].getfirstr()+ii*drk
                       - Barbies[j].getfirstr() - jj*drb) <= 1 &&
              Math.abs (Kens[i].getfirstc()+ii*dck
                       - Barbies[j].getfirstc() - jj*dcb) <= 1;
     }
   }
   if (!touch) kwb++;
  }
  out.printf ("There are %d Ken(s) without Barbie.\r\n",kwb);
 }

 /* This method, given a start position and direction finds
    the word of the specified length */
 public static String ExtractWord (int r, int c, int dr, int dc,
                                   int len) throws Exception {

  String word = "";
  for (int i=0; i < len; i++)
   word += ExtractChar (r+i*dr,c+i*dc);
  return word;
 }

 /* This builds the array of Tuples finding every instance of the
    desired string.  The number of found strings is returned in sz. */
 public static Tuple[] GetString (String s,int[] sz) throws Exception{

  Tuple[] stuff = new Tuple[8*rsz*csz];
  int ct = 0;
  for (int r=0; r<rsz; r++)
   for (int c=0; c<csz; c++) //Try all eight directions
    for (int dr = -1; dr <= 1; dr++)
     for (int dc = -1; dc <= 1; dc++)
      if (dr !=0 || dc != 0) {
       String word = ExtractWord (r,c,dr,dc,s.length());
       if (word.equals(s)) { //We found a word, add it
        Tuple t = new Tuple (r,c,r+dr*(s.length()-1),
                             c+dc*(s.length()-1));
        stuff[ct++] = t;
       }
      }
  sz[0] = ct;
  return stuff;
 }

 /* This method extracts a single character from the Grid, but
    returns a space if the position is out of bounds. */
 public static char ExtractChar (int r, int c) throws Exception {

  if (r>=0 && r < rsz && c>=0 && c < csz) return Grid[r][c];
  return ' ';
 }
}

/* This class keeps track of the positions of the first and last
   characters of a word */
class Tuple {

 private int firstr,firstc,lastr,lastc;
 public int getfirstr () {return firstr;}
 public int getfirstc () {return firstc;}
 public int getlastr () {return lastr;}
 public int getlastc () {return lastc;}
 public Tuple (int fr, int fc, int lr, int lc) {
  firstr = fr;
  firstc = fc;
  lastr = lr;
  lastc = lc;
 }
}


```