Strings

```
/**
    * Tests to see if the String is a palindrome using recursion
    */
   public static boolean isPalindrome(String str)
   {
      if( str.length() <= 1 )
         return true;

      if( str.charAt(0) == str.charAt(str.length()-1) )
         return isPalindrome( str.substring(1, str.length()-1) );
      else
         return false;
   }



/**
    *  tests if str1 equal to str2 except for one character
    */
     public static boolean one (String str1, String str2) {
      if (str1.equals("") || str2.equals(""))
         return false;
      if (str1.charAt(0) == str2.charAt(0))
         return one(str1.substring(1), str2.substring(1));
       else
         return (str1.substring(1)).equals(str2.substring(1));
      }
/**
```

```
/**
    * Tests to see if the String is an anagram using recursion
    */
public static boolean isAnagram (String s1,String s2)

   {

     if(s1.length()!=s2.length())
          return false;
      if(s1.length()==0)
           return true;
     int place= s2.indexOf(s1.charAt(0));
    if(place<0)
        return false;
    else
     {
         String a=s2.substring(0,place);
         String b=s2.substring(place+1);
         return isAnagram(s1.substring(1),a.concat(b));



   }

}
```

```
*  Prints all binary numbers of length n
    */
   public static void binaryNumbers(int n) {
      binaryNumbers(n, "");
   }

   private static void binaryNumbers(int n, String str) {
      if (n == 0) {
         System.out.println(str);
      }
      else {
         binaryNumbers( n-1, str+"0" );
         binaryNumbers( n-1, str+"1" );
      }
   }
```

נגדיר: ג'וקר הוא תחליף למחרוזת תווים כלשהי (ריקה, בעלת תו אחד או יותר). נייצג ג'וקר על-ידי התו כוכבית ('*').

נגדיר: שתי מחרוזות, s1 ו-s2, הן **דומות-תבנית** אם הן זהות, כאשר המחרוזת s2 עשויה להכיל ג'וקרים (אחד או יותר או כלל לא).

דוגמאות:

- המחרוזות "TheExamIsEasy" ו- "The*xamIs*y" **הן** דומות-תבנית.
- המחרוזות "TheExamIsEasy" ו- "Th*mIsEasy*" **הן** דומות-תבנית.
- המחרוזות "TheExamIsEasy" ו- "*" **הן** דומות-תבנית.
- המחרוזות "TheExamIsEasy" ו- " TheExamIsEasy" **הן** דומות-תבנית.
- המחרוזות "TheExamIsEasy" ו- " The*IsHard" **אינן** דומות-תבנית.

שימו לב: תוי הג'וקר ('*') יכולים להופיע במחרוזת השניה בלבד! וכן, לא יתכן שיהיו שני ג'וקרים צמודים ב- s2. (כלומר, s2 לא יכולה להיות, למשל, "ab**c"). **אתם יכולים להניח זאת ואינכם צריכים לבדוק.**

כתבו שיטה סטטית **רקורסיבית** samePattern שחתימתה היא:

```
public static boolean samePattern (String s1, String s2)
```

המקבלת שתי מחרוזות s1 ו- s2 , ומחזירה true אם המחרוזות דומות-תבנית, ו- false אחרת.

השיטה צריכה להיות רקורסיבית ללא שימוש בלולאות כלל.

שימו לב, בפתרון הבעיה מותר להשתמש אך ורק בשיטות שלהלן (המוגדרות במחלקה String), ובפקודות השוואה.

- `public char charAt(int i)` - המחזירה את התו במקום ה- i במחרוזת (עליה היא מופעלת)
- `public String substring(int i)` - המחזירה את הסיפא המתחילה במקום ה- i במחרוזת עליה היא מופעלת. כלומר, את התת-מחרוזת מהמקום ה-i עד לסוף המחרוזת. לדוגמא, אם `s = "abc"` אז `s.substring(1)` יחזיר את `"bc"`.
- `public String substring(int i, int j)` - המחזירה את התת-מחרוזת המתחילה במקום ה- i ומסתיימת במקום ה-j (לא כולל) במחרוזת עליה היא מופעלת. לדוגמא, אם `s = "abcdefg"` אז `s.substring(2,5)` יחזיר את `"cde"`.
- `public int length()` - המחזירה את אורך המחרוזת עליה היא מופעלת.

```
public static boolean samePattern(String s1, String s2) {
        // base case
        if (s1.length()==0) {
            if (s2.length()==0 || s2.equals("*"))
                return true;
            else
                return false;
        }
        else if (s2.length()==0)
                return false;
        // rec calling
        if (s1.charAt(0)==s2.charAt(0))
            return samePattern(s1.substring(1), s2.substring(1));
        if ('*'==s2.charAt(0))
            return samePattern(s1.substring(1), s2) || samePattern(s1, s2.substring(1));
        else
            return false;
    }
```

Arrays

```
/**
    * move negative numbers and 0 to beginning of array, positive
    * numbers to end of array
    */
public static void negpos(int a[])
{
 negpos(a,0,a.length-1);
}

private static void negpos(int a[],int h, int t){
    if(h>=t)
         return;
     if (a[h]<=0)
          negpos(a,h+1,t);
     else if(a[t]>0)
           negpos(a,h,t-1);
     else
     {
        int temp=a[h];
        a[h]=a[t];
        a[t]=temp;
        negpos(a,h+1,t-1);
      }
    }
```

```
/**
    * calculate sum of  the array
    */

private static int sum(int a[], int n){
     if(n==0)
        return a[0];
     else
        return a[n]+ sum(a,n-1);
       }
```

```
/**
    * Tests  if all array components larger than sum of  those before
    */
 public static boolean allBiggerSum(int a[])
  {
     return allBiggerSum( a,a.length-1);
    }
  private static boolean allBiggerSum(int a[],int n)
  {
      if (n==0)
          return true;
     if(sum(a,n-1)>=a[n])
          return false;
      return allBiggerSum(a,n-1);
    }
```

אפשר יותר יעיל , תחשבו כיצד

```
/**
    * calculate number in array. each digit in different cell followed by -1 till the end
of the array
    */
public static int calc(int a[]){
     return calc(a,a.length-1);
    }
   private static int calc(int a[],int n){
          if (n==0)
             return(a[n]);
           if(a[n]==-1)
                return calc(a,n-1);
            return calc(a, n-1)*10+a[n];
           }
```

```
public class rec{
    public static boolean cover (int [] values, int amount){
        return cover(values, 0,amount);
 }

        private static boolean cover (int [] values ,int i,
int amount)  {
     if (i==values.length)
      return false;
     if (values[i] == amount)
      return true;
     return (cover(values,i+1,amount) ||
cover(values,i+1,amount - values[i]));

    }
public static void main(){
   int[] values = {5, 22, 13, 5, 7, -4};
   System.out.println(cover(values,42));
   System.out.println(cover(values,31));
   System.out.println(cover(values,17));
   System.out.println(cover(values,13));
   System.out.println(cover(values,-5));
}}
```

```
/**
     * calculate sum of diagonal
     */

public static int sumDiagonal(int a[][]){
      return sumDiagonal(a,a.length-1);
    }
  private static int sumDiagonal(int a[][],int n){
       if(n==0)
            return a[0][0];
       return a[n][n]+  sumDiagonal(a,n-1);
    }
```

```
    /**
     * calculate sum of all square above main  diagonal
     */
    public static int sumAbove(int a[][]){
      return sumAbove(a,a.length-1);
    }
   private static int sumAbove(int a[][],int n){
       if(n==-1)
            return 0;
       return sumLine(a,n,n)+  sumAbove(a,n-1);
    }
    private static int sumLine(int a[][],int line,int col){
        if(col==a[line].length)
                return 0;
         return sumLine(a,line,col+1)+a[line][col];
        }



public static int sumAbove2(int a[][]){
      return sumAbove2(a,a.length-1,a.length-1);
    }
   private static int sumAbove2(int a[][],int i,int j ){
       if(i==-1)
            return 0;
       if (j==a.length)
          return sumAbove2(a,i-1,i-1);
       return sumAbove2(a,i,j+1)+a[i][j];
    }
```

```java
public static boolean find(int mat[][], int x) {
        return find(mat, x, 0, 0, mat.length);
}
public static boolean find(int mat[][], int x, int startRow, int startCol, int size) {
        int smallest = mat[startRow][startCol];
        int biggest = mat[startRow+size-1][startCol+size-1];
        if (x < smallest || x > biggest) return false;
        if (size > 1) {
                int half = size/2;
                return find(mat, x, startRow,         startCol,      half) ||
                        find(mat, x, startRow+half,   startCol,      half) ||
                        find(mat, x, startRow,        startCol+half, half) ||
                        find(mat, x, startRow+half,   startCol+half, half);
                }
        return mat[startRow][startCol] == x;
}
```