unas soluciones

TRES Y UN POQUITO MÁS EN KOTLIN

```
fun main() {
  val casos = readLine()!!.toInt()
  for(i in 1..casos){
     val linea= readln().split(' ')
     val p=linea[0].toDouble()
     val d=linea[1].toDouble()

     val posiblePI = p / d
     if (posiblePI >= 3.1415 && posiblePI <= 3.1417) {
        println("ES CIRCUNFERENCIA")

     } else {
        println("PARECE UN HUEVO")
     }

  }
}
```

DIBUJAR PERÍMETRO DE UN CUADRADO

```
static void cuadrado(int lado, char c){
      for(int linea=0;linea<lado;linea++){
         if(linea==0 || linea==lado-1){// para imprimir primera y última
            for(int posicion=0;posicion<lado;posicion++){
               System.out.print(c+" ");
            }
            System.out.println("");
         }else{ // para líneas intermedias
            for(int posicion=0;posicion<lado;posicion++){
               if(posicion==0 || posicion==lado-1 ){
                  System.out.print(c+" ");//
               }else{
                  System.out.print("  "); //2 espacios;
               }

            }
            System.out.println("");
         }
      }
}
```

PICOS Y VALLES

```
import java.util.Scanner;
public class Unidad3 {

   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      int numAlturas = sc.nextInt();

      while (numAlturas != 0) {
         int[] alturas = new int[numAlturas];
         for (int i = 0; i < alturas.length; i++) {
            alturas[i] = sc.nextInt();
```

```
        }
        int picos=0;
        int valles=0;
        for (int i = 0; i < alturas.length; i++) {
            int anterior=i==0?alturas.length-1:i-1;
            int siguiente= i==alturas.length-1?0:i+1;
            if(alturas[i]>alturas[anterior]&&alturas[i]>alturas[siguiente]){
                picos++;
            }
            if(alturas[i]<alturas[anterior]&&alturas[i]<alturas[siguiente]){
                valles++;
            }

        }
        System.out.println(picos+valles);
        numAlturas = sc.nextInt();
    }

  }
}
```

Veni, vidi, vici(cifrado césar)

SOLUCIÓN BASADA EN ARITMÉTICA DE CHAR
esta solución se basa en los códigos unicode y al trabajar con char hay que tener cuidado con detalles de aritmética char por ejemplo:

- ojo con las expresiones tipo c+=d; hay que tener claro que es equivalente a c =(char) (c+d);
- hay que tener mucho cuidado con las expresiones y operadores por ejemplo observar que

  'Z'- ('A'-c-1)
  es lo mismo que
  'Z'- ('A'-c)+1)

- y por supuesto cuidado con los castings

```
static String cesar(String text, int d) {
    String cifrado = "";
    text = text.toUpperCase();
    for(int i=0; i<text.length(); i++) {
        char c = text.charAt(i);
        if(c>='A' && c<='Z') {//si  es letra la ciframos
            c =(char) (c+d);

            //despues de sumar d podría  ocurrir que me pase por derecha o izquierda
            if(c>'Z'){ //me pasé por derecha
                //me pongo en A y me desplazo c-'Z'-1 veces. resto 1 porque ponerse en z ya es 1 desplazamiento
                c = (char) ('A'+ (c-'Z'-1));
            } else if(c<'A' ){//me pasé por izquierda
                //me pongo en Z y de desplazo a la izquierda 'A' -c -1 veces
                 c = (char) ('Z'-  ('A'-c-1) );
            }
        }
        cifrado += c;
    }
    return cifrado;

  }
```

La solución también se puede basar en indexof y un alfabeto cualquiera y se puede hacer más compleja si queremos que trabaje para valores de *d* no necesariamente **-26<=d<=26**
pero con lo anterior es más que suficiente en este caso

MYSCANNER

el test que mira que no uses scanner además implica que si lo pasas escribiste el armazón de todos los métodos y lo puntuó con el doble que las otras(mark 2.0), aunque como hay tantos casos a penas se nota.

```
__tester__.mirarSiUsaScanner();
```

```
OK no hay clase Scanner
```

Use as example | Display: Mostrar | Ocultar el resto si falla | Mark: 2 | Ordering: 10


```
class MyScanner {
    private int pos;
    private String datos;
    MyScanner(String s){
        this.datos=s;
        this.pos=0;
    }

    int getPos(){return pos;}
    String getDatos(){return datos;}

    boolean  hasNextLine(){
        return pos<datos.length()?true:false;
    }
    String nextLine(){
        String linea="";
        while(pos<datos.length()){
            char c=datos.charAt(pos);
            pos++;
            if(c=='\n')
                break;
            linea=linea+""+c;

        }
        return linea;
    }
    boolean  hasNext(){
        //si hay token quiere decir que hay algun caracter no delimitador entre pos y datos.length()

        boolean hayToken=false;
        if(pos<datos.length()){
            String sinAnalizar=datos.substring(pos);
            for(int i=0;i<sinAnalizar.length();i++){
                char c=sinAnalizar.charAt(i);
                if(c!='\n' && c!='\t' && c!=' '){
                    hayToken=true;
                    break;
                }

            }

        }
        return hayToken;
    }
    String next(){

        String token="";
        char c=' ';
        //primer bucle para posicionarse en el primer carácter no delimitador
        while(pos<datos.length()){
            c=datos.charAt(pos);
```

```
            if(c=='\n' || c=='\t' || c==' ')
             pos++;
            else
              break;
        }

        //en este punto o bien  llegué al final del string o tengo el primer caracter válido del token
        if(pos<datos.length()){ //si no llegué a final del String incluyo caracter en token
            token=token+""+c;
            pos++;
        }
        //seguimos construyendo token si no llegamos a final
        while(pos<datos.length()){
            c=datos.charAt(pos);
            if(c=='\n' || c=='\t' || c==' ') //fin de token
                break;
            token=token+""+c;
            pos++;
        }
        return token;
    }
    int nextInt(){
        return Integer.parseInt(next());
    }

}
```