Examen de convocatoria ordinaria, 12 de enero de 2024. Fundamentos de Programación.
Grado en Ingeiería Informática y Grado e Estadística.
Propuesta de soluciones:

Ejercicio 1

Los tres primeros apartados, comunes a T1 y T2-T3. El cuarto (d) solamente para T1:

a) (3 versiones)

```
public static boolean consecutivas3V1 (char v[]) {
	// DEVUELVE si el vector tiene o no tres oes o tres equis consecutivas
	// PRE: v no es nulo
	// Método: comodón: transforma el vector en cadena de caracteres
	//                  (poco generalizable)
	String linea = "";
	for (int i=0; i<v.length; i++) linea += v[i];
	return (
			linea.indexOf("xxx")>= 0
			||
			linea.indexOf("ooo")>= 0
			);
}

public static boolean consecutivas3V2 (char v[]) {
	// DEVUELVE si el vector tiene o no tres oes o tres equis consecutivas
	// PRE: v no es nulo
	// Método: comparar cada elemento a considerar con los dos anteriores
	boolean consecutivas = false;
	int i = 2;  // comienza en el tercer elemento
	while (i<v.length && !consecutivas) {
		if (v[i] == 'o' || v[i] =='x')
			if (v[i] == v[i-1] && v[i] == v[i-2]) consecutivas = true;
		i++;
	}
	return consecutivas;

}

public static boolean consecutivas3V3 (char v[]) {
	// DEVUELVE si el vector tiene o no tres oes o tres equis consecutivas
	// PRE: v no es nulo y los valores de sus casillas son 'o', 'x' o '-'
	// Método: contar número de caracteres consecutivos iguales no '-'

	int i = 0;
	char primero = v[0];
	int contIguales = 1;

	while (contIguales != 3 && i<v.length-1) {
		i=i+1;
		if (primero == '-') {
			primero = v[i];
			contIguales = 1;
		} else if (v[i] == primero) {
			contIguales ++;
		} else {
			primero  = v[i];
			contIguales = 1;
		}
	}
	return contIguales == 3;

}
```

b)

```java
public static boolean filaConsecutivas3 (char[][]m, int n) {
	/* DEVUELVE si la matriz tiene o no tres oes o tres equis consecutivas
	 *                en la fila n
	 * PRE: m no es nula
	 *        y  0 <= n < m.length (existe tal fila)
	 */
	return consecutivas3V2(m[n]);
}
```

c)

```java
public static boolean columnaConsecutivas3 (char[][]m, int n) {
	/* DEVUELVE si la matriz tiene o no tres oes o tres equis consecutivas
	 *                en la columna n
	 * PRE: m no es nula
	 *        y  0<= n <m[0].length
	 */
	char v[] = new char [m.length];
	for (int i=0; i<m.length; i++) {
		v[i] = m[i][n];
	}
	return consecutivas3V2(v);
}
```

d)

```java
public static double sumaDiag(double[][] mat, int n) {
	/*
	 * Entrada (parámetros):
	 *                matriz de números mat
	 *                un entero (número de columna)
	 * Devuelve: la suma de los elementos de la matriz
	 *                de una línea inclinada -45%
	 *                comenzando en el elemento n de la primera fila
	 * Observaciones: los índices de fila y columna comienzan en 0
	 * PRE: matriz no nula y 0 <= n < mat[0].length
	 *
	 */
	int nfilas = mat.length, ncolumnas = mat[0].length;
	double suma = 0;
	int posi = 0, posj = n;
	while (posi < nfilas && posj <ncolumnas) {
		suma += mat[posi][posj];
		posi++;
		posj++;
	}
	return suma;
}
```

Ejercicio 2 de T1, 4 de T2-T3

```java
public static void imprimeDinosaurio(int n) {
	// PRE n>=0
	if (n>0) {
		imprimeDinosaurio (n-1);
		System.out.print ( " " + n + " ");
		imprimeDinosaurio (n-1);
	}
}
```

Ejercicio 3 de T1, 2 de T2-T3

```
public class Ej3 {

    public static void main(String[] args) {
      /*
       * Entrada: por teclado, sucesión de palabras consituídas por minúsculas
       *                (termina por palabra que no cumpla esa condición)
       *                y por cada una de ellas un entero para codificarla
       * Salida:  fichero de texto con la sucesión de claves-palabras
       *                codificadas con e cifrado de César
       * Observaciones: si la entrada es la cadena vacía
       *                         el programa seguirá esperando cadena no vacía
       * Algoritmo: bucle controlado por centinela
       *            (palabra con algún carácter no minúscula)
       */

      final  int INF = -65, SUP = 65;
      Scanner in = new Scanner (System.in);
      String palabra;
      int clave;

      try {
            PrintWriter f = new PrintWriter(new FileWriter ("cifrado.txt"));

            System.out.print ("Palabra: ");
            palabra = in.nextLine();      // leer palabra (posible centinela)

            while (todoMinusculas (palabra)) { // control por centinela

                  // obtener clave
                  System.out.print ("Clave (no nula y en [-65,65]:) ");
                  clave = in.nextInt();
                  while (clave <INF || clave > SUP || clave == 0) {
                        System.out.print ("No es válida. ");
                        System.out.print ("Clave (no nula y en [-65,65]: ");
                        clave = in.nextInt();
                  }

                  // codificar y escribir
                  f.println (clave + " " + codificada (palabra, clave));

                  // leer siguiente palabra
                  System.out.print ("Palabra: ");
                  palabra = in.next();
            }

            System.out.println ("Proceso terminado");
            f.close();

      } catch(IOException e) {
            System.out.println ("No se ha podido");
      }
    }
```

```
    public static String codificada(String palabra, int clave) {
      // PRE: palabra y clave garantizan que la codificación no se sale de rango
      //          (ver enunciado)
      //  NOTA: concretamente funciona bajo la precondición siguiente
      //                aunque esto no se pedía en el examen
      //          mínimo código ASCII de palabra + clave >= 0 y
      //          máximo código ASCII de palabra + clave <= 255
      String nueva = "";
      for (int i = 0; i<palabra.length(); i++)
            nueva += (char) (palabra.charAt(i)+clave);
      return nueva;
    }

    public static boolean todoMinusculas(String palabra) {
      // Devuelve 'true' si 'palabra' está formada por
      //          minúsculas del alfabeto latino
      // Obs: a la cadena vacía devuelve 'false'
      boolean vamosBien;
      if (palabra.length()==0) vamosBien = false;
      else { // palabra no vacia
            int i = 0;
            vamosBien = true;
            while (vamosBien && i<palabra.length()) {
                  char c = palabra.charAt(i);
                  if (c<'a' || c>'z') vamosBien = false;
                  else i++;
            }
      }
      return vamosBien;
    }
}
```

Ejercicio 4 de T1, 3 de T2-T3

a)

```
class regCIFRADO {
      String cadena;
      int clave;
      boolean cifrada; // true = cifrada, false = original

      public regCIFRADO(String cadena, int clave, boolean cifrada) {
            this.cadena = cadena;
            this.clave = clave;
            this.cifrada = cifrada;
      }

      public regCIFRADO() {
            this.cadena = "";
            this.clave = 0;
            this.cifrada = false;
      }
}
```

b)

```
public static void cambioSiClave (regCIFRADO[] v, int clave) {
	// PRE: clave correcta y cadenas correctas...
	for (int i = 0; i < v.length; i++)
		if (v[i].clave == clave) {
			if (v[i].cifrada)
				v[i].cadena = codificada(v[i].cadena, -clave);
			else
				v[i].cadena = codificada(v[i].cadena, clave);
			v[i].cifrada = ! v[i].cifrada;
		}
}
```

Ejercicio 5 de T2-T3

```
public static Nodo eliminado(Nodo list, int pos) {
	/*
	 * Si existe un nodo en la posición pos, lo elimina
	 * si no, deja la lista como está.
	 *
	 * Se empieza a contar por 1
	 *
	 * PRE: pos >= 1
	 * DEVUELVE : la lista modificada (o no)
	 */

	if (list != null) {
		if (pos == 1)
			list = list.sgte;
		else { // pos >=2
			Nodo anterior = list;
			int contador = 2;
			while (anterior.sgte != null && contador < pos) {
				contador++;
				anterior = anterior.sgte;
			}
			// anterior.sgte = null y anterior señala al último nodo
			// ó
			// contador = pos y anterior es previo al que hay que borrar
			if (anterior.sgte != null)
				anterior.sgte = anterior.sgte.sgte;
		}

	}
	return list;
}
```