PSP UT 1 EXAMEN FEBRERO EJERCICIO 5 (igual que palíndromo)

Un programa compila y lanza a otro...

Clase HolaMundo

```
package PSPReto5;

public class HolaMundo {
   public static void main(String[]args){
      System.out.println("Hola Mundo");
   }
}
```

Clase Lanzadora

```
package PSPReto5;

import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.util.Scanner;
import java.util.logging.Level;
import java.util.logging.Logger;


public class Lanzadora {
   public static void main(String[] args) {
      try {
         ProcessBuilder pb2 = new ProcessBuilder("javac", "HolaMundo.java");
         Process p2 = pb2.start();

        /* try {
            p2.waitFor();
         } catch (InterruptedException ex) {
            Logger.getLogger(Ejercicio7.class.getName()).log(Level.SEVERE, null, ex);
         }*/

         ProcessBuilder pb1 = new ProcessBuilder("java", "HolaMundo");
         Process p1 = pb1.start();

         int exitVal;
         try {
            //El proceso actual espera hasta que el subproceso Process finalice
            exitVal = p1.waitFor(); //Recoge la salida de System.exit()
            if (exitVal == 0) {
               System.out.println("Salida del repetidor");
               System.out.println("====================");
            } else {
               System.out.println("MALLLLLLLLL");
            }
         } catch (InterruptedException e) {
            e.printStackTrace();
         }
         try {
            InputStream is = p1.getInputStream();
            int c;
            while ((c = is.read()) != -1) {
               System.out.print((char) c);
            }
            is.close();
         } catch (Exception e) {
            e.printStackTrace();
         }

         //Redirección a un archivo
         File f = new File("SalidaEjercicio7.txt");
         pb1.redirectOutput(f).start();

      } catch (IOException ex) {
         System.out.println("Falló");
      }
   }
}
```

PSP UT 1 EXAMEN FEBRERO EJERCICIO 11 (sin enunciado)

Ni idea a qué viene este ejercicio. Incluye Búsqueda de Patron.

Clase ProgReto11

```
package ProgReto11;

public class PruebaReto11 {
   public static void main(String[] args){
      Molecula m1=null;
      try{
         m1 = new Molecula("H2O");
      }catch(Exception e){
         System.out.println("Error al instanciar el objeto");
      }
      System.out.println("El número de átomos distintos es "+m1.getNumAtomosDistintos());
      System.out.println("La cantidad de átomos en la molécula es de "+m1.getNumAtomosTotales());
      System.out.println(m1.toString());


      Molecula m2=null;
      try{
         m2 = new Molecula("H02O");
      }catch(Exception e){
         System.out.println("Error al instanciar el objeto");
      }

      Molecula m3=null;
      try{
         m3 = new Molecula("2O");
      }catch(Exception e){
         System.out.println("Error al instanciar el objeto");
      }
   }
}
```

Clase Molecula

```
package ProgReto11;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Molecula {
   private String[] atomos;
   private int[]cantidades;
   private int numElementos;

   public Molecula(String formula){
      this.atomos = new String[10];
      this.cantidades =  new int[10];
      this.numElementos=0;

      //Generar patrón
      String expresion = "([A-Z][a-z]?([2-9]|[1-9][0-9])*)+";
      Pattern patron = Pattern.compile(expresion);
      Matcher acoplamiento = patron.matcher(formula);
      String aux;
      boolean patronEncontrado; // Si el patron ha sido encontrado
      int inicio,fin;
      int numVeces= 0; // Contador para saber cuántas veces aparece el patrón
      while(acoplamiento.matches()) {
            atomos[numElementos] = acoplamiento.group(1);
            cantidades[numElementos] = Integer.parseInt(acoplamiento.group(2));
            numElementos++;
      }
   }
```

```
    public String[] getListaAtomos() {
        String[] aux = new String[numElementos];
        for(int i=0; i<numElementos;i++){
            aux[i]=atomos[i];
        }
        return aux;
    }

    public int getAtomo(String atomo){
        int pos=-1;
        for(int i=0; i<numElementos && pos==-1;i++){
            if(atomos[i].equals(atomo)){
                pos = i;
            }
        }
        if(pos != -1){
            //Estoy aquí dentro pq he encontrado el átomo
            return cantidades[pos];
        }else{
            //Estoy aquí dentro pq NO he encontrado el átomo
            return 0;
        }
    }

    public int getNumAtomosDistintos(){
        return this.numElementos;
    }

    public int getNumAtomosTotales(){
        int sumatoria=0;
        for(int i=0;  i<numElementos; i++){
            sumatoria+=cantidades[i];
        }
        return sumatoria;
    }

    public String toString(){
        String salida="";
        for(int i=0; i<numElementos;i++){
            salida = salida +atomos[i];
            if(cantidades[i]!=1)
                salida = salida + cantidades[i];
        }
        return salida;
    }
}
```

PSP UT 3 EXAMEN FEBRERO EJERCICIO Preparación Servidores

EJERCICIOS TIPOS EXAMEN
2.- Examen febrero 2021 - Ejercicio 4. [2 puntos] Servidor de tiempo.

Implementar un programa **servidor TCP** llamado *Ejercicio04.java* que escuche por el **puerto 2021** y que sólo sea capaz atender a un único cliente (servidor no concurrente). Este servidor tan solo tendrá que entender un comando: la palabra "***TIME***". Cuando reciba este comando desde un cliente, el servidor devolverá la **hora local del servidor** en el formato que tú prefieras.

El resultado en pantalla de la ejecución del programa *Ejercicio04* debería ser el siguiente:

SERVIDOR DE TIEMPO

Servidor de Profesor
Servidor iniciado.
Escuchando por el puerto 2021.
Esperando conexión con cliente.

En tu caso deberá aparecer tu nombre en lugar de la palabra "*Profesor*". Y así se quedará "esperando" hasta que algún cliente le llegue a solicitar la hora actual a través del comando "*TIME*". Cuando finalmente reciba este comando, se le enviará a la hora local del servidor al cliente que la haya solicitado y el programa servidor finalizará su ejecución ordenadamente. En caso de recibir cualquier otro tipo de comando o petición que no sea "TIME" se deja a opción del alumnado que decida qué hacer (enviar un mensaje de error, finalizar, no hacer nada, etc.).

Clase Cliente (Falta la parte en la que el cliente recibe la info aunque el servidor la manda..)

```
public class clienteTiempo {
    public static void main(String[] args) throws IOException {

        String ip = "localhost";
        int puerto = 2021;
        Socket sCliente = null;
        String opcionesPalabra[] = {"PERRETE", "TIME", "HOLA", "ADIOS", "FUEGO"};
        String palabraEnviada = null;

        sCliente = new Socket(ip, puerto);
        OutputStream salida = sCliente.getOutputStream();
        DataOutputStream flujoSalida = new DataOutputStream(salida);
        do {
            palabraEnviada = opcionesPalabra[(int) (Math.random() * opcionesPalabra.length)];
            flujoSalida.writeUTF(palabraEnviada);
        } while (!palabraEnviada.equals("TIME"));

        System.out.println("Palabra TIME enviada al servidor");
    }
}
```

Clase Servidor

```
public static void main(String[] args) throws IOException {
        // TODO code application logic here
        int puerto = 2021;
        ServerSocket skServidor = null;
        Socket cliente = null;

        InputStream entrada = null;
        DataInputStream flujoEntrada = null;
        OutputStream salida = null;
        DataOutputStream flujoSalida = null;

        skServidor = new ServerSocket(puerto);
        System.out.println("Esperando conexión del cliente....");
        cliente = skServidor.accept();
        System.out.println("conexión aceptada!");

        entrada = cliente.getInputStream();
        flujoEntrada = new DataInputStream(entrada);
        salida = cliente.getOutputStream();
        flujoSalida = new DataOutputStream(salida);
        String datosEntrada = new String();

        datosEntrada = flujoEntrada.readUTF();
        while (!datosEntrada.equals("TIME")) {
           System.out.println("Comando no reconocido. La palabra recibida es: " + datosEntrada);
           datosEntrada = flujoEntrada.readUTF();

        }
        System.out.println("Palabra: "+datosEntrada+ " recibida");
        DateFormat dateFormat = new SimpleDateFormat("HH:mm:ss");
        Date date = new Date();
        System.out.println("Hora actual: " + dateFormat.format(date));
        flujoEntrada.close();
        flujoSalida.close();
        cliente.close();
        skServidor.close();
    }
}
```

PSP UT 3 EXAMEN FEBRERO EJERCICIO Preparación Servidores

EJERCICIOS TIPOS EXAMEN

3.- Examen febrero 2021 - Ejercicio 5. [2 puntos] Cliente de tiempo.

Implementar un programa cliente TCP llamado *Ejercicio05.java* que envíe una solicitud de tiempo (comando "*TIME*") al posible servidor de tiempo que haya escuchando en el puerto 2021 de la propia máquina en la que se ejecute el cliente. El programa debería mostrar por pantalla la información recibida desde el servidor.

Si había algún servidor escuchando en el puerto y se recibe una respuesta, el programa debería mostrar la siguiente información por pantalla:

CLIENTE DE TIEMPO

Cliente de Profesor
Conectándose a localhost por el puerto 2021.
Solicitando información del tiempo local.
Información recibida del servidor de tiempo: TIME: 17:41:38
En tu caso deberá aparecer tu nombre en lugar de la palabra "*Profesor*"

Si por el contrario, no había ningún servidor escuchando por el puerto especificado, el resultado debería ser este otro:

CLIENTE DE TIEMPO

Cliente de Profesor
Conectándose a localhost por el puerto 2021.
Error: No se ha podido establecer conexión con el servidor.
En ambos casos, el programa finalizaría su ejecución.