República bolivariana de Venezuela

Ministerio del poder popular para la educación

Universidad Rafael Urdaneta

PROF: PAOLA RODRÍGUEZ

Catedra: Ingeniería Del software

Principios de la Programación.

Integrantes:

Anderson Gutierrez 29758990

Tomas Sandoval 25194896

Principio YAGNI (*You aren't gonna need it*).

Ejemplo: Una empresa pide hacer un chat sencillo que solo envié mensajes para poder hablar entre el grupo de trabajo, debido que solo es para establecer un dialogo sin necesidad de enviar más nada no se necesitaría de ninguna otra funcionalidad.

Método que envía el mensaje.

```
public class envioMensaje  implements ActionListener{

    @Override
    public void actionPerformed(ActionEvent e) {

        try {

            Socket conexion = new Socket("192.168.0.107", 3000);

            paquete_Envio envio_datos = new paquete_Envio();
            envio_datos.setNick(nick.getText());
            envio_datos.setIp(ip.getSelectedItem().toString());
            envio_datos.setMensaje(mensaje.getText());

            ObjectOutputStream paquete_datos = new ObjectOutputStream(conexion.getOutputStream());
            paquete_datos.writeObject(envio_datos);

            calcularFecha();
            charla2.append("\n"+nick.getText()+":"+mensaje.getText());
            calcularHora();
            charla2.append("\n-------------------------------------------------------------------------------------------\n");

            conexion.close();

        } catch (UnknownHostException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        } catch (IOException e1) {
            // TODO Auto-generated catch block
            System.out.println(e1.getMessage());
        }
        Borrar();
    }
}
```

Y al programador se le acurre hacer que el chat sea más funcional para el futuro haciéndolo envía archivos de cualquier tipo (.txt, .jpg, etc).

El detalle aquí es que se está casi seguro de que no será utilizada la funcionalidad añadida y al aplicar el principio YAGCI la respuesta de qué hacer con esa funcionalidad extra es:

Crear una funcionalidad que **no se va a usar es tiempo perdido*, es decir no se utilizara***.

Nota: debido a que este chat es una aplicación cliente-servido-cliente, lo que hace que si la funcionalidad extra no será utilizada habrá mucho código sin implementar tanto del lado del cliente como del servidor.

Nota: La utilidad extra lleva mucho más código que la que se necesita y como las probabilidades de uso son casi nulas se debería de obviar.

Método que se encarga de buscar y enviar el archivo.

```
public void buscar_enviar_archivo(Socket envio, File abrir,File file) throws IOException{
    Thread hilo1 = new Thread(this);
    hilo1.start();

        try{
           //------Metodo que permite cargar la ventana--------//
            archivo = new JFileChooser();
            archivo.showOpenDialog(this);
            //-------------Abrimos El Archivos Seleccionado-----------------//
            abrir = archivo.getSelectedFile();
            envio = new Socket("192.168.0.107",9090);
            //Creamos el archivo que vamos a enviar
            file = new File(abrir.getPath());

            //Obtenemos el tamaño
            int tamanoArchivo = 0;
            tamanoArchivo = (int)file.length();
            System.out.println("Tamaño archivo (Cliente): "+tamanoArchivo);

            //Creamos un fluio de salida
            DataOutputStream salida = new DataOutputStream(envio.getOutputStream());
            System.out.println("Enviando archivo: "+file.getName());
            salida.writeUTF(file.getName());

            //Creamos un fluio de entrada
            FileInputStream entrada = new FileInputStream(abrir.getPath());
            BufferedInputStream leer = new BufferedInputStream(entrada);

            //Creamos un fluio de salida para enviarlo en byte
            BufferedOutputStream salida_byte = new BufferedOutputStream(envio.getOutputStream());

            // Creamos un array de tipo byte con el tamaño del archivo
            byte[] cantidad = new byte[0];
            cantidad =  new byte[tamanoArchivo];
            // Leemos el archivo
            int linea;
            while((linea = leer.read( cantidad )) >= 0){
                salida_byte.write(cantidad,0,linea);
            }

            System.out.println( "Archivo Enviado: "+file.getName());

            //cerramos todo
            leer.close();salida_byte.close();envio.close();
        }catch(Exception ex1){System.out.println(ex1.getMessage());
        }
}
```

Solución: La solución en este caso es simplemente eliminar la funcionalidad extra para que la aplicación sea mucho más eficiente adaptándose a las necesidades de la empresa. Quedando solo en método que se encarga de enviar los mensajes.

```
public class envioMensaje  implements ActionListener{

    @Override
    public void actionPerformed(ActionEvent e) {

        try {

            Socket conexion = new Socket("192.168.0.107", 3000);

            paquete_Envio envio_datos = new paquete_Envio();
            envio_datos.setNick(nick.getText());
            envio_datos.setIp(ip.getSelectedItem().toString());
            envio_datos.setMensaje(mensaje.getText());

            ObjectOutputStream paquete_datos = new ObjectOutputStream(conexion.getOutputStream());
            paquete_datos.writeObject(envio_datos);

            calcularFecha();
            charla2.append("\n"+nick.getText()+":"+mensaje.getText());
            calcularHora();
            charla2.append("\n-----------------------------------------------------------------------------------------------\n");

            conexion.close();

        } catch (UnknownHostException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        } catch (IOException e1) {
            // TODO Auto-generated catch block
            System.out.println(e1.getMessage());
        }
        Borrar();
    }
}
```

Nota: Debido a que el programador ha invertido tiempo en este método lo recomendable seria guardar el código fuente y si en un futuro se necesita se implementa.

Nota General: solo se muestra código de la parte del cliente para que la explicación sea breve.

Dry code

Aquí tenemos una sección de un ejemplo de una página de registro de usuario HTML:

```
</head>

   <body>

      <main class="register-container">

         <section class="form-container">

            <h1 class="tittle">REGISTRO</h1>

            <form method="post" action="Register" id="register-form">



               <div class="content-name">

               <label class="form-label" for="name">Name:</label>

               <input class="form-input" id="name" type="text" placeholder="name" name="name"
required><br>

               </div>

               <div class="content-email">

               <label class="form-label" for="email">Email:</label>

               <input class="form-input" id="email" type="email" placeholder="Example@gmail.com"
name="email" required><br>

               </div>
```

```
                <div class="content-password">

                <label class="form-label" for="pass">Password:</label>

                <input class="form-input" id="password" type="password" placeholder="password"
name="password" required>

                </div>

               <div>
```

Ahora esta sección de CSS trata de englobar los elementos content-name , content-email y content-password por separado:

```
.content-name{

    padding:15px;

    background-color: #23A6D5;

    text-align: center;

    margin-left: 10px;

}

.content-email{

    padding:15px;

    background-color: #23A6D5;

    text-align: center;

    margin-left: 10px;

}
```

```
.content-password{

    padding:15px;

    background-color: #23A6D5;

    text-align: center;

    margin-left: 10px;

}
```

El código CSS se repite para poder brindarle los mismos atributos a cada contenedor, una forma de simplificar esto sería, cambiando todas las clases content-name content-email y content-password a una sola clase “content” que contenga las 3 clases previas. Visualmente más limpio para el programador de esta manera:

```
.content {

    padding:15px;

    background-color: #23A6D5;

    text-align: center;

    margin-left: 10px;

}
```

Ahora si cada content tuviera información personal y otra en común, se crearía una clase content global para abarcarlas todas, y una segunda con su typeo personal ejemplo content-name, como mostramos previamente. Y si se quieren incluir los mismos atributos a ambas clases solo se señalan de esta manera:

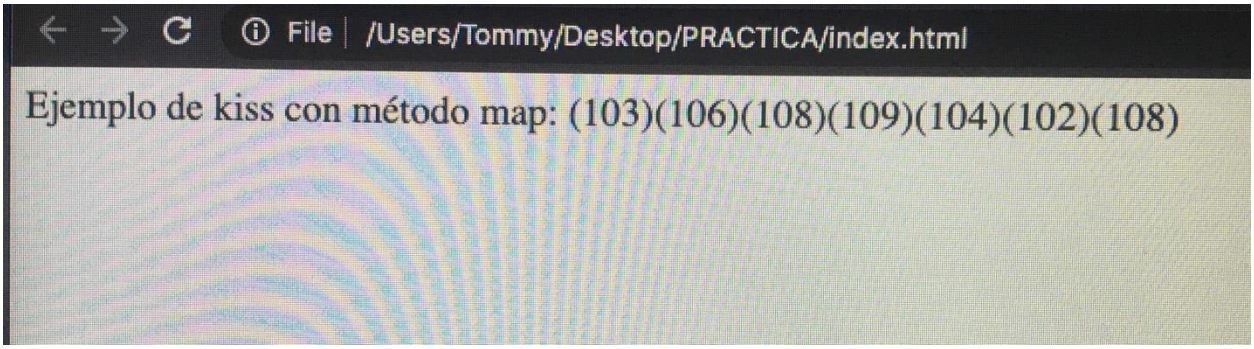
Content Content-name{ } . En el css.

Kiss Code

Con un ejemplo de javascript, teniendo una variable de arreglo de números (3,6,8,9,4,2,8) llamada Arraydenumeros que se requiere sumarle 100 a cada variable del arreglo, se debe aplicar el método a cada variable por separado, lo cuál abarcarían muchas lineas de código, una manera de simplificarlo es usando el método “map” el cuál le aplica el método de sumarle 100 a cada variable en el arreglo en una sola función con i:

```
<!DOCTYPE html>

<html>

<body>

   Ejemplo de kiss con método map:

   <script type="text/javascript">

      var Arraydenumeros =[3,6,8,9,4,2,8];

      var suma= Arraydenumeros.map(x => x+100);

      for(i=0;i<suma.length;i++){

         document.write("("+suma[i]+")");

      }

   </script>

</body>

</html>
```

Dando como resultado:



Ejemplo De los 3 principios.

Nos piden hacer un proyecto simple que haga las operaciones matemáticas básicas simultáneamente con los mismos números:

```
function operacion_suma (x,y){
    return x+y;
}
function operacion_resta (x,y){
    return x-y;
}
function operacion_multiplicacion (x,y){
    return x*y;
}
function operacion_division (x,y){
    return x/y;
}

function operacion_modulo (x,y){
    return x%y;
}
```

Al programador se le ocurrió ponerle un método más al no saber si podría llegar a usar en un futuro.

Solución: Aplicando el principio YAGNI se eliminaría el método que no nos piden, luego aplicando KISS se podría reducir a solo método y se haría mas simple, y aplicando el DRY se evitaría la duplicidad de código y se haría un método llamado “Operaciones” que las englobaría todas las operaciones.

```
function Operaciones (x,y){
    var suma, resta, multiplicacion, division;

    suma = x+y;
    console.log("Suma= "+suma);

    resta = x-y;
    console.log("resta= "+resta);

    multiplicacion = x*y;
    console.log("multiplicacion= "+multiplicacion);

    division = x/y;
    console.log("division= "+division);
}
```