Trabajo Práctico N° 5: Refactoring

1) ¿Cuál de los siguientes son "bad smell code"?

a) el nombre de la clase es muy largo
b) el método tiene muchos parámetros
c) en la clase Punto hay un atributo que se llama "x".
d) en la clase Animal hay un atributo que se llama "x".
e) la clase es "public"
f) el método retorna "void"
g) la subclase redefine todos los métodos de la super clase y no tiene métodos distintivos.
h) el método es muy largo
g) el bucle while tiene una condición muy compleja.

2) Identificar Bad Code Smell en el siguiente segmento de código, aplicar el/los refactoring apropiados.

```
void calcularDemanda() {
      COSTO= getSaldoCliente()*1.24% - _MAX;
      //imprime los detalles del cliente
      System.out.println ("Nombre:   " + NAME);
      System.out.println ("Apellido: " + APELLIDO);
      System.out.println ("Cuenta:   " + getCuentaCliente());
      System.out.println ("Costo:    " + COSTO);
}
```

3) Identificar "bad Smell code" y mejorar el código aplicando los refactoring necesarios.

a)

```
boolean buscoElemento(int valor)
{
   int i;
   boolean enc;
   enc=false;
   i=0;

   while (i<actual && enc==false)
   {
         if (elementos[i]==valor)
         enc=true;
         i++;
   }
         return enc;
```

b)

```
boolean verificarPalindromo (String cadena)
{
      int i, dim;
      Pilachar aux;
      aux = new Pilachar();
      OperacionesPilaChar aux2;
      aux2= new OperacionesPilaChar();

      for (i=0;i<cadena.length()/2;i++)
            aux.meterChar(cadena.charAt(i));
      if (cadena.length()%2!=0)
            dim=i+1;
      else
            dim=i;
      while(!aux.pilaVacia() && palin==true)
      {
             letra=aux.sacarChar();
```

```
                if (letra==cadena.charAt(dim))
                      dim++;
                else
                      palin=false;
           }
            if (palin==false)
                  return false;
            else
                  return true;
      }
```

c)

```
        class Empresa {
          pubblic String nombre;
          private String mail;
          private int codTelPais;
          private int codTelCiudad;
          private int codtelempresa;

          int getCodTelPais( ) {..}
          void setCodTelPais(int ) {..}
          int getCodTelCiudad( ) {..}
          void setCodTelCiudad(int ) {..}
          int getCodTelEmpresa( ) {..}
          void setCodTelEmpresa(int ) {..}
          String getTelefono() {
           return
              String.valueOf(codTelCiudad)+String.valueOf(codTelEmpresa)+
              String.valueOf(codTelPais);
          }
          String getMail( ) {..}
          void setMail(String ) {..}
          String getNombre( ) {..}
          void setNombre(String ) {..}
        }
```

d)

```
        public class Dinero {
            public int x;
            public String tipoMoneda;

            Dinero(int cantidad, String tipoMoneda) {
                x = cantidad;
                this.tipoMoneda=tipoMoneda;  }

            public float convertirAPesos {     {
               float aux = 0;
               if (tipoMoneda.equals("DÓLAR"))
                      aux = x * 3.70;
               if (tipoMoneda.equals("EURO"))
                      aux = x * 5,10;
```

```
                if (tipoMoneda.equals("REAL"))
                        aux = x * 1,85;
                if (tipoMoneda.equals("PESOS"))
                        aux = x;
                return aux;
            }
        }
```

4) El siguiente código realiza la encriptación de una secuencia de caracteres. Identifique los "code bad smell" y aplique los refactoring de manera de obtener un código de mejor calidad.

```
import java.io.*; //libreria de entrada / salida
import java.lang.*; //Libreria del lenguaje

public class encripta {//Nombre de la clase

        public static void main (String[] args){
                //Variables utilizadas
                String lsdato_entrada="";
                String lsdato_salida ="";
                char lc_comodin;

                System.out.println("Introduzca los datos a encriptar:");
                //Aqui se cargan los datos a encriptar
                 try
                 {
                    InputStreamReader isr = new InputStreamReader(System.in);
                    BufferedReader texto = new BufferedReader(isr);
                    lsdato_entrada=texto.readLine();
                 }
                 catch(IOException e) //Muestra si hay errores al cargar los datos
                 {
                    System.err.println("Errors:" + e.getMessage());
                 }
                //Aquí empieza la encriptación por método ASCII, al valor de cada letra
                //Le suma el valor de su posición y lo va concatenando en una nueva
                //variable
                for (int i=0;i<=lsdato_entrada.length()-1;i++)
                {
                        lc_comodin=lsdato_entrada.charAt(i);
                        for (int i2=0;i2<=lsdato_salida.length();i2++)
                        {
                                lc_comodin++;
                        }
                        lsdato_salida=lsdato_salida+lc_comodin;
                }
                //Despliega el resultado encriptado
                System.out.println(lsdato_salida);
                //Inicializa la variable para desencriptar
                lsdato_entrada = lsdato_salida;
                lsdato_salida = "";
                //Aquí desencripta utilizando el mismo concepto, pero le resta en lugar
                //de sumar para obtener el valor original
                for (int i=0;i<=lsdato_entrada.length()-1;i++)
                {
                        lc_comodin=lsdato_entrada.charAt(i);
                        for (int i2=0;i2<=lsdato_salida.length();i2++)
                        {
                                lc_comodin--;
                        }
                        lsdato_salida=lsdato_salida+lc_comodin;
                }
                //Muestra la cadena desencriptada
                System.out.println(lsdato_salida);
        }
}
```

5) Identificar los bad smell code en la clase TATETI y aplicar los refactoring necesarios para obtener un mejor código.

```
import java.io.*;

class tateti{
	public static void main(String [] args) {
	int m=0 ,n=0 ,ch2=1;

	char matrix[][];//={0};
	matrix = new char[3][3];
	while(ch2==1){
		for (m=0;m<3;m++)for (n=0;n<3;n++)matrix[m][n]= '\0';
		int i,j,sum=0;
		while ( sum < 10)
		{
		 if (sum == 0)
		 {
			System.out.println (" ");
			System.out.println ("                 1   2   3");
			System.out.println ("              1  "+matrix[0][0]+" |
				"+matrix[0][1]+" | "+matrix[0][2]);
			System.out.println ("                ---|---|---");
			System.out.println ("              2
				"+matrix[1][0]+" | "+matrix[1][1]+" |
				"+matrix[1][2]);
			System.out.println ("                ---|---|---");
			System.out.println ("              3  "+matrix[2][0]+" |
				"+matrix[2][1]+" | "+matrix[2][2]);
		 }
		System.out.println ("Jugador 1 es 'X': selecciona
					renglon y columna");
		System.out.print ("Renglon : ");
		i = Console.readInt();
		System.out.print("Columna : ");
		j = Console.readInt();
		for (;i>3 || i<1 || j>3 || j<1 ||('X'==matrix[i-1][j-
				1]||'O'==matrix[i-1][j-1]);)
		   System.out.println ("Lo siento, escoge otro
				renglon y columna.");
		   System.out.print ("Renglon : ");
		   i=Console.readInt();
		   System.out.print ("columna : ");
		   j=Console.readInt();
		}
		matrix[i-1][j-1]='X';
		sum++;
		System.out.println (" ");
		System.out.println ("                   1   2   3");
		System.out.println ("                1  "+matrix[0][0]+" | "+matrix[0][1]+"
			| "+matrix[0][2]);
		System.out.println ("                  ---|---|---");
		System.out.println ("                2  "+matrix[1][0]+" | "+matrix[1][1]+"
			| "+matrix[1][2]);
		System.out.println ("                  ---|---|---");
		System.out.println ("                3  "+matrix[2][0]+" | "+matrix[2][1]+"
			| "+matrix[2][2]);

		if (matrix[0][0]=='X' && matrix[0][0]==matrix[1][1] &&
			matrix[1][1]==matrix[2][2])
			{System.out.println ("Jugador 1 gana");break;}
		if (matrix[2][0]=='X' && matrix[2][0]==matrix[1][1] &&
			matrix[1][1]==matrix[0][2])
			{System.out.println ("Jugador 1 gana");break;}

		if (matrix[0][0]=='X' && matrix[0][0]==matrix[1][0] &&
			matrix[1][0]==matrix[2][0])
			{System.out.println ("Jugador 1 gana");break;}
		if (matrix[0][1]=='X' && matrix[0][1]==matrix[1][1] &&
			matrix[1][1]==matrix[2][1])
			{System.out.println ("Jugador 1 gana");break;}
		if (matrix[0][2]=='X' && matrix[0][2]==matrix[1][2] &&
			matrix[1][2]==matrix[2][2])
			{System.out.println ("Jugador 1 gana");break;}
		if (matrix[0][0]=='X' && matrix[0][0]==matrix[0][1] &&
			matrix[0][1]==matrix[0][2])
```

```
                {System.out.println ("Jugador 1 gana");break;}
            if (matrix[1][0]=='X' && matrix[1][0]==matrix[1][1] &&
                matrix[1][1]==matrix[1][2])
                {System.out.println ("Jugador 1 gana");break;}
            if (matrix[2][0]=='X' && matrix[2][0]==matrix[2][1] &&
                matrix[2][1]==matrix[2][2])
                {System.out.println ("Jugador 1 gana");break;}
            if (sum == 9){
                System.out.println ("Nadie gana..."); break;}

            //Turno del jugador 2

            System.out.println ("Jugador 2 es 'O': selecciona renglon y columna");
            System.out.print ("Renglon : ");
            i = Console.readInt();
            System.out.print ("Columna : ");
            j = Console.readInt();

            for (;i>3 || i<1 || j>3 || j<1 ||('X'==matrix[i-1][j-1]||'O'==matrix[i-
                1][j-1]);)
                {System.out.println ("Debes escoger otra posicion");
                System.out.print ("Renglon : ");
                i = Console.readInt();
                System.out.print ("Columna : ");
                j = Console.readInt();}

            matrix[i-1][j-1]='O';
            sum++;

            System.out.println (" ");
            System.out.println ("                1   2   3");
            System.out.println ("              1  "+matrix[0][0]+" | "+matrix[0][1]+"
                | "+matrix[0][2]);
            System.out.println ("                ---|---|---");
            System.out.println ("              2  "+matrix[1][0]+" | "+matrix[1][1]+"
                | "+matrix[1][2]);
            System.out.println ("                ---|---|---");
            System.out.println ("              3  "+matrix[2][0]+" | "+matrix[2][1]+"
                | "+matrix[2][2]);

            if (matrix[0][0]=='O' && matrix[0][0]==matrix[1][1] &&
                matrix[1][1]==matrix[2][2])
                {System.out.println ("Jugador 2 gana");break;}
            if (matrix[2][0]=='O' && matrix[2][0]==matrix[1][1] &&
                matrix[1][1]==matrix[0][2])
                {System.out.println ("Jugador 2 gana");break;}
            if (matrix[0][0]=='O' && matrix[0][0]==matrix[1][0] &&
                matrix[1][0]==matrix[2][0])
                {System.out.println ("Jugador 2 gana");break;}
            if (matrix[0][1]=='O' && matrix[0][1]==matrix[1][1] &&
                matrix[1][1]==matrix[2][1])
                {System.out.println ("Jugador 2 gana");break;}
            if (matrix[0][2]=='O' && matrix[0][2]==matrix[1][2] &&
                matrix[1][2]==matrix[2][2])
                {System.out.println ("Jugador 2 gana");break;}
            if (matrix[0][0]=='O' && matrix[0][0]==matrix[0][1] &&
                matrix[0][1]==matrix[0][2])
                {System.out.println ("Jugador 2 gana");break;}
            if (matrix[1][0]=='O' && matrix[1][0]==matrix[1][1] &&
                matrix[1][1]==matrix[1][2])
                {System.out.println ("Jugador 2 gana");break;}
            if (matrix[2][0]=='O' && matrix[2][0]==matrix[2][1] &&
                matrix[2][1]==matrix[2][2])
                {System.out.println ("Jugador 2 gana");break;}

            } // 2do while
        System.out.println (" ");
        System.out.println ("Desea jugar otra vez??? 1=si 2=no");
        ch2 = Console.readInt();
        } // 1er while
        } // main
} // class
```

6) Identificar los refactoring automatizados por Eclipse. (*vistos en clase*)

Refactoring	S/ N	Refactoring	S/N
(110) Extract Method		(238) Decompose Conditional	
(117) Inline Method		(240) Consolidate Conditional Expression	
(119) Inline Temp		(243) Consolidate Duplicate Conditional Fragments	
(120) Replace Temp with Query		(245) Remove Control Flag	
(124) Introduce Explaining Variable		(250) Replace Nested Conditional with Guard Clauses	
(128) Split Temporary variable		(255) Replace Conditional with Polymorphism	
(131) Remove Assignments to Parameters		(260) Introduce Null Object	
(135) Replace method to method object		(267) Introduce Assertion	
(139) Substitute algorithm		(273) Rename Method	
(142) Move Method		(275) Add Parameter	
(146) Move Field		(277) Remove Parameter	
(149) Extract Class		(279) Separate Query from Modifier	
(154) Inline Class		(283) Parameterize Method	
(157) Hide Delegate		(285) Replace Parameter with Explicit Methods	
(160) Remove Middle Man		(288) Preserve Whole Object	
(162) Introduce Foreign Method		(292) Replace Parameter with Method	
(164) Introduce Local Extension		(295) Introduce Parameter Object	
(171) Self Encapsulate Field		(300) Remove Setting Method	
(175) Replace Data Value with object		(303) Hide Method	
(179) Change Reference to Value		(304) Replace Constructor with Factory Method	
(186) Replace Array with Object		(308) Encapsulate Downcast	
(189) Duplicate Observed Data		(310) Replace Error Code with Exception	
(197) Change Unidirectional Association to Bidirectional		(315) Replace Exception with Test	
(200) Change Bidirectional Association to Unidirectional		(320) Pull Up Field	
(204) Replace Magic Number with Symbolic Constant		(322) Pull Up Method	
(206) Encapsulate Field		(325) Pull Up Constructor Body	
(208) Encapsulate collection		(328) Push Down Method	
(217) Replace Record with Data Class		(329) Push Down Field	
(218) Replace type code with class		(330) Extract Subclass	
(223) Replace type code with subclass		(336) Extract Superclass	
(227) Replace type code with state/strategy		(341) Extract Interface	
(232) Replace subclass with fields		(344) Collapse Hierarchy	

(352) Replace Inheritance with Delegation		(345) Form Template Method	
(355) Replace Delegation with Inheritance			

7) Identificar los refactoring automatizados por Eclipse (*no vistos en clase*)

Nombre	**Función**	**SI/NO**
Rename	Renombra el elemento seleccionado y corrige todas las referencias al elemento, también en otros archivos. Por ejemplo métodos, parámetros de métodos, atributos, variables, constantes, paquetes, proyectos.	
Move	Mueve el elemento seleccionado y corrige todas las referencias al elemento, también en otros archivos. Aplicable para clases, paquetes, proyectos. Eclipse además permite mover métodos o atributos estáticos.	
Change Method Signature	Cambia los nombres de los parámetros, los tipos, el orden de los mismos y actualiza todas las referencias al correspondiente método. También se puede agregar o remover parámetros, cambiar el tipo de retorno o la visibilidad del mismo. NetBeans solo permite el cambio de los parámetros y la visibilidad del mismo.	
Convert Anonymous Class to Nested	Ayuda a convertir una clase anónima anidada a una clase miembro.	
Move Member Class to New File / Move Inner to Outer Level	Crea una nueva clase con esta clase miembro y actualiza todas las referencias. Un atributo es agregado a la nueva clase para permitir el acceso, si es necesario, a la clase que antes la contenía.	
Push Down / Push Members Down	Mueve un conjunto de métodos y atributos de una clase a la subclase.	
Pull Up / Pull Members Up	Mueve un conjunto de métodos y atributos declarados en una clase a la superclase.	
Extract Interface	Crea una nueva interface con los métodos seleccionados y la clase que la implementa. Si es posible actualiza las referencias a la nueva interface.	
Extract Superclass	Crea una superclase basada en una clase existente.	
Use SuperClass Where Possible	Reemplaza si es posible las ocurrencias de una clase con su superclase.	
Inline	Pone en línea una variable local, método o constante.	
Extract Method / Introduce Method	Crea un nuevo método que contiene las sentencias o expresión seleccionada y reemplaza la selección con una referencia la nuevo método.	
Extract Local Variable / Introduce Variable	Crea una nueva variable asignándole la expresión seleccionada y reemplaza la selección con la referencia a la nueva variable.	
Extract Constant / Introduce Constant	Crea una constante a partir del elemento seleccionado y reemplaza la ocurrencia de la selección con la nueva constante.	
Introduce Factory	Crea un método Factory que llama al constructor seleccionado y retorna el objeto creado. Todas las referencias a este constructor son reemplazadas por llamadas al nuevo método Factory.	
Introduce Parameter	Reemplaza una expresión con una referencia a un nuevo parámetros del método y actualiza todas las llamadas al método para poder pasarle la expresión como un valor del parámetro.	
Convert Local Variable to Field / Introduce Field	Cambia una variable local en un atributo de la clase.	
Encapsulate Field	Reemplaza todas las referencias a un atributo creando los métodos get y set.	

Move Class	Mueve una clase a otro paquete o adentro de otra clase y todo el proyecto es actualizado a la nueva ubicación de la clase.	
Generify Refactoring	Analiza y transforma el código para que use tipos genéricos.	
Replace Inheritance With Delegation	Permite remover una clase de una jerarquía de herencia, mientras preserva la funcionalidad del padre.	