	<p style="text-align: center;">UNIVERSIDAD DON BOSCO FACULTAD DE ESTUDIOS TECNOLÓGICOS</p>
<p style="text-align: center;">CICLO I</p>	<p style="text-align: center;">GUIA DE LABORATORIO #2 Programación Orientada a Objetos Estructuras de decisión control y colecciones</p>

I. OBJETIVOS.

- Implementar los conceptos fundamentales de sentencias de control.
- Crear programas que se vean reducidos en código al utilizar sentencias repetitivas.

II. INTRODUCCIÓN.

Estructuras de control

Todo lenguaje de programación cuenta (o al menos debería contar), con una serie de instrucciones que le permitan controlar el flujo de ejecución. Java afortunadamente posee dos grandes rubros de dichas sentencias:

Estructuras selectivas, en las cuales encontramos el if, else-if y switch.

Estructuras repetitivas o ciclos, entre las cuales están while, do... while y for.

Estructuras selectivas:

En la vida muchas veces es necesario elegir entre un camino y el otro a seguir. En muchas de las actividades que realizamos día con día, nos enfrentamos a decisiones que debemos tomar y que de una u otra forma alteran el cauce normal de nuestra vida (o de nuestro programa).

Como otros lenguajes, Java cuenta con estructuras para el control de flujo. Las decisiones se pueden realizar simples o anidadas(es decir, varias condiciones una dentro de otra).

Sentencia if:

if/else evalúa una condición y determina el curso a seguir; si la condición es falsa, se ejecuta el bloque **else..**

La instrucción **if** tiene la siguiente estructura:

```
if (condición)  
{  
    //Código a ejecutar si condición es true  
}  
else  
{  
    //Código a ejecutar si condición es false  
}
```

Esta instrucción evalúa la expresión condición, y si es true, ejecuta el código que hay entre las llaves que hay debajo de if. Si condición fuese false, el código a ejecutar sería el contenido entre las llaves que existen debajo de else.

La parte else es opcional, es decir, esto también es correcto:

```
if (condición)  
{  
    //Código a ejecutar si condición es true  
}
```

En este caso si condición es false no sucede nada, la instrucción no ejecuta ninguna instrucción. Otra simplificación también correcta es que en caso de que sólo exista una instrucción dentro de las llaves (del if, o del else) se pueden eliminar las llaves, es decir:

```
if (condición) //instrucción a ejecutar si condición es true  
else //instrucción a ejecutar si condición es false
```

Lo que no está permitido eliminar en ningún caso, son los puntos y coma de las instrucciones que empleemos en la parte verdadera (if) o falsa (else), tengamos o no las llaves.

Tome en cuenta

Operadores Relacionales: Permiten comparar variables según relación de igualdad/desigualdad o relación mayor/menor. Devuelven siempre un valor boolean.

- '>': Mayor que
- '<': Menor que
- '==': Iguales
- '!=': Distintos
- '>=': Mayor o igual que
- '<=': Menor o igual que

Operadores Lógicos: Nos permiten construir expresiones lógicas.

- '&&' : devuelve true si ambos operandos son true.
- '||' : devuelve true si alguno de los operandos es true.
- '!' : Niega el operando que se le pasa.
- '&' : devuelve true si ambos operandos son true, evaluándolos ambos.
- '|' : devuelve true uno de los operandos es true, evaluándolos ambos.

Sentencia switch:

switch /case. Permite seleccionar una opción de un conjunto. Si la variable indicada en **switch** coincide con la condición **case**, el bloque es ejecutado. Adicionalmente, la instrucción **break** finaliza el bloque de código case.

También existen ocasiones o programas donde se exige evaluar muchas condiciones a la vez, en estos casos, o se usa una condición compuesta muy grande o se debe intentar convertir el problema a uno que se pueda resolver usando la instrucción `switch()`;

La instrucción `switch()` es una instrucción de decisión múltiple, donde el compilador prueba o busca el valor contenido en una variable contra una lista de constantes ints o chars, cuando el computador encuentra el valor de igualdad entre variable y constante, entonces ejecuta el grupo de instrucciones asociados a dicha constante, si no encuentra el valor de igualdad entre variable y constante, entonces ejecuta un grupo de instrucciones asociados a un default, aunque este último es opcional.

Nota: Desde JDK 7, podemos usar una cadena literal/constante para controlar una declaración **switch**, lo cual no es posible en C/C++. Usar un modificador basado en cadena/string es una mejora con respecto al uso de la secuencia equivalente `if/else`.

El formato de esta instrucción es el siguiente:

Capturar o asignar variable de condición;

```
switch(var int o char)
{
    case const1: instrucción(es);
        break;
    case const2: instrucción(es);
        break;
    case const3: instrucción(es);
        break; .....
    default: instrucción(es);
};
```

Estructuras iterativas:

Los cálculos simples o la manipulación de pequeños conjuntos de datos se pueden realizar fácilmente a mano, pero las tareas grandes o repetitivas son realizadas con mayor eficacia por una computadora, ya que estas están especialmente preparadas para ello.

Para repetir varias veces un proceso determinado haremos uso de los ciclos repetitivos, a los cuales se les conoce con el nombre de estructura repetitiva, estructuras iterativas, lazo o bucles.

En C, al igual que en Java podemos encontrar tres tipos de ciclos.

- Entrada Asegurada (while).

```
while ( condición ) {
    conjuntoDeSentencias
}
```

- Ciclo Controlado por Contador (for)

```
for (inicialización, expresionBooleana, incremento) {
    conjuntoDeSentencias;
}
```

- Hacer Mientras (do.. while)

```
do{
    conjuntoDeSentencias;
```

```
}  
while ( condición )
```

Colecciones en Java

Una colección es un objeto que puede almacenar un número variable de elementos siendo cada elemento otro objeto.

Java tiene desde la versión 1.2 todo un juego de clases e interfaces para guardar colecciones de objetos. En él, todas las entidades conceptuales están representadas por interfaces, y las clases se usan para proveer implementaciones de esas interfaces.

Una interfaz nos dice qué podemos hacer con un objeto. Un objeto que cumple determinada interfaz es “algo con lo que puedo hacer X”. La interfaz no es la descripción entera del objeto, solamente un mínimo de funcionalidad con la que debe cumplir. Java organiza todas las clases de colecciones bajo el paquete `java.util`

- **Collection**

Una Collection es todo aquello que se puede recorrer (o iterar) y de lo que se puede saber el tamaño. Muchas otras clases extenderán la interfaz Collection imponiendo más restricciones y dando más funcionalidades. La sintaxis para usar un collection es la siguiente:

```
Collection <clase_elementos> nombre_objeto = new Collection<clase_elementos> ();
```

- **List**

La interfaz List define una sucesión de elementos donde cada uno de ellos lleva un índice asociado. Así, podríamos tener una lista con los nombres de las personas que han utilizado un servicio de acceso a internet que podría ser: usuarios --> (Jorge López, Andrea Maldonado, Luis Pérez, Elías Sánchez, Mirna Acosta). Donde cada contenido va asociado a un índice, `usuario(0)` sería Jorge López, `usuario(1)` sería Andrea Maldonado, `usuario(2)` sería Luis Pérez y así sucesivamente. Sintaxis:

```
List <clase_elementos> nombre_objeto = new List<clase_elementos> ();
```

- **Map**

Un Map representa lo que en otros lenguajes se conoce como “diccionario” y que se suele asociar a la idea de “tabla hash” (aunque no se implemente necesariamente con esa técnica). Un Map es un conjunto de valores, con el detalle de que cada uno de estos valores tiene un objeto extra asociado. A los primeros se los llama “claves” o “keys”, ya que son los que permiten acceder a los segundos.

Un Map no es una Collection ya que esa interfaz es limitada. Podríamos decir que Collection es unidimensional, mientras que un Map es bidimensional. Su sintaxis es la siguiente:

```
Map <clase_claves, clase_valores> nombre_objeto = new Map< clase_claves, clase_valores > ();
```

III. PROCEDIMIENTO.

Sentencia if

Ejemplo 1

Los siguientes programas nos permite utilizar la sentencias condicional if, digitar cada uno de ellos y si tiene alguna duda consultar a su docente.

- Nombre de la clase **Controlif**.

```
import java.util.*;
public class Controlif {
    public static void main(String[] args) {
        Scanner reader = new Scanner(System.in);
        int Var1,Var2;
        System.out.print("Ingrese numero1: ");
        Var1=reader.nextInt();
        System.out.print("Ingrese numero2: ");
        Var2=reader.nextInt();
        if(Var1==Var2){
            System.out.print("Los numeros ingresados son iguales ");
        }
        else{
            System.out.println("Los numeros ingresados No son iguales ");
        }
    }
}
```

Ejemplo 2

- Nombre de la clase **ControlIf2**

```
import javax.swing.JOptionPane;
public class ControlIf2 {
    public static void main(String[] args) {
        String v1="";
        v1=JOptionPane.showInputDialog("Ingrese un valor");
        if(v1==null)
        {
            JOptionPane.showMessageDialog(null,"Apretaste cancelar");
        }else {
            if(v1.equals(""))
            {
                JOptionPane.showMessageDialog(null,"No ingresaste nada");
            }else{

```

```

        JOptionPane.showMessageDialog(null,"El valor de V1 es: "+v1);
    }
}
}
}

```

Ejemplo 3

- Nombre de la clase **Controllf3**

```

import javax.swing.JOptionPane;
public class Controllf3 {
    public static void main(String[] args) {
        int result = JOptionPane.showConfirmDialog(null,"Replace existing selection?");
        System.out.println("El numero devuelto es: "+result);

        if (result == JOptionPane.YES_OPTION) {
            System.out.println("Yes");
        } else if (result == JOptionPane.NO_OPTION) {
            System.out.println("No");
        } else if (result == JOptionPane.CANCEL_OPTION) {
            System.out.println("Cancel");
        } else if (result == JOptionPane.CLOSED_OPTION) {
            System.out.println("Closed");
        }
    }
}

```

Ejemplo 4

Diseñe un programa en java que calcule los descuentos a un trabajador sabiendo que son aplicables un 6.25% del salario en AFP solo si este es superior a \$300.00. Además, si es un trabajador hombre se le descuenta aparte del AFP, el 3% sobre el sueldo en conceptos de ISSS y 10% en concepto de RENTA.

- Nombre de la clase **SentenciaIF**.

```

import javax.swing.*;
public class SentenciaIF {
    public static void main(String[] args) {
        String datos;
        double sueldo, afp=0, iss, totalre, nsueldo, renta;
        datos = JOptionPane.showInputDialog(" Ingrese el sueldo del empleado (a) ");
        sueldo = Double.parseDouble(datos);
        if(sueldo>300){
            afp = (sueldo*0.0625);
        }
        JOptionPane.showMessageDialog(null, " Este empleado tiene un sueldo de " + sueldo + " y el descuento del AFP es "+ afp);
        datos= (String) JOptionPane.showInputDialog(null,

```

```

"Ingresa el Sexo:\nSi es Masculino (M)\nSi es Femenino (F)",
"Sexo del Empleado",JOptionPane.QUESTION_MESSAGE,
null, //Icono por defecto
new Object[] { "M", "F"},
"F"); //opcion por defecto
if(datos == "M")//Si es de sexo Masculino
{
    isss = sueldo*0.03;
    renta=sueldo*0.10;
    totalre = afp+isss+renta;
    nsueldo = sueldo - totalre;
    JOptionPane.showMessageDialog(null,"A este empleado se le detiene" + isss+ "en
concepto
de ISSS\nAdemas se le retiene: "+ renta+ "En concepto de Renta\nLo que hace un total de "+
totalre+ "\nY su nuevo Sueldo es de:"+ nsueldo);
}
}
}

```

Sentencia switch

Ejemplo 5

- Nombre de la clase **Switch1**.

```

public class Switch1 {
    public static void main(String[] args) {
        int a = 1;
        int b = 1;
        char op = '+'; // Puede cambiar el operador aritmético...
        System.out.print("El resultado es : ");
        switch ( op ) {
            case '+':
                System.out.println( a + b );
                break;
            case '-':
                System.out.println( a - b );
                break;
            case '*':
                System.out.println( a * b );
                break;
            case '/':
                System.out.println( a / b );
                break;
            default:
                System.out.println("error" );
        }
    }
}

```


Ejemplo 6

- Nombre de la clase **Switch2**.

```
import javax.swing.JOptionPane;
public class Switch2 {
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        int replaced = JOptionPane.showConfirmDialog(null, "Replace existing selection?");
        String result = "?";
        switch (replaced) {
            case JOptionPane.CANCEL_OPTION:
                result = "Canceled";
                break;
            case JOptionPane.CLOSED_OPTION:
                result = "Closed";
                break;
            case JOptionPane.NO_OPTION:
                result = "No";
                break;
            case JOptionPane.YES_OPTION:
                result = "Yes";
                break;
            default:
                ;
        }
        System.out.println("Replace? " + result);
    }
}
```

Ejemplo 7

- Para este ejemplo se utilizara una variable String que desde la versión 7 de la JDK es soportada.
- Nombre de la clase **StringSwitchDemo**.

```
public class StringSwitchDemo {
    public static int getMonthNumber(String month) {
        int monthNumber = 0;

        if (month == null) {
            return monthNumber;
        }

        switch (month.toLowerCase()) {
            case "january":
                monthNumber = 1;
                break;
            case "february":
```

```
        monthNumber = 2;
        break;
    case "march":
        monthNumber = 3;
        break;
    case "april":
        monthNumber = 4;
        break;
    case "may":
        monthNumber = 5;
        break;
    case "june":
        monthNumber = 6;
        break;
    case "july":
        monthNumber = 7;
        break;
    case "august":
        monthNumber = 8;
        break;
    case "september":
        monthNumber = 9;
        break;
    case "october":
        monthNumber = 10;
        break;
    case "november":
        monthNumber = 11;
        break;
    case "december":
        monthNumber = 12;
        break;
    default:
        monthNumber = 0;
        break;
}

return monthNumber;
}

public static void main(String[] args) {

    String month = "february";

    int returnedMonthNumber =
        StringSwitchDemo.getMonthNumber(month);

    if (returnedMonthNumber == 0) {
        System.out.println("Invalid month");
    } else {
        System.out.println(returnedMonthNumber);
    }
}
```

```
}  
}
```

Bucle for

Ejemplo 8

- Nombre de la clase **For**.

```
import javax.swing.JOptionPane;  
public class For {  
    public static void main(String[] args) {  
        // TODO code application logic here  
        int numero;  
        int valor;  
        String strnumero = JOptionPane.showInputDialog("Ingrese un Numero: ");  
        numero=Integer.parseInt(strnumero);  
        valor=numero;  
        //long resultado=1;  
        for(int i=1; i<valor; i++){  
            numero=numero*i;  
            System.out.println(numero);  
        }  
        JOptionPane.showMessageDialog(null,"El factorial de "+valor+" es: "  
+numero,"Respuesta",JOptionPane.INFORMATION_MESSAGE);  
        System.out.println("El factorial es "+numero);  
    }  
}
```

Bucle For Mejorado

Otra de las novedades que tiene Java 5.0 es el ciclo for mejorado. Este nuevo bucle es para recorrer colecciones con una sintaxis más simple. Veamos su sintaxis:

```
for(tipo idParam : colección){  
    Sentencias;  
}
```

- Necesitamos la colección por la cual iterar.
- El tipo no es más que la clase de los elementos que contiene esta colección.
- Finalmente necesitaremos también un nombre de parámetro (idParam) que representa el elemento de la colección que se va extrayendo en cada iteración.

Ejemplo 9

Crear una nueva clase con el nombre **NuevoFor** y agregar el siguiente código.

```
public class NuevoFor {  
  
    /**  
     * @param args the command line arguments
```

```

*/
public static void main(String[] args) {
    // TODO code application logic here
    int Numero[]={1, 2, 3, 4, 5, 6, 7, 8, 9, 0};
    int Suma=0;
    //for clasico
    for(int indice=0; indice<10; indice++){
Suma=Suma + Numero[indice];
    }
    System.out.println("Total con for clasico " + Suma);
    Suma=0;
    System.out.println("Suma reinicializada vale " + Suma);
    //for mejorado
    for(int valor: Numero){
        Suma += valor;
    }
    System.out.println("Total con for mejorado " + Suma);
}
}

```

Bucle while

Ejemplo 10

En una empresa, se tienen datos correspondientes a los sueldos de 5 empleados, de los cuales, se desea saber, quien goza del sueldo mayor, quien goza del sueldo menor y cuantos poseen un sueldo mayor de \$300.00.

- Nombre de la clase **SentenciaWhile**.

```

import javax.swing.*;
public class SentenciaWhile {
    public static void main(String[] args)
    {
        // TODO code application logic here
        String leer;
        double sueldo, mayor=0,menor=10000;
        int i=1, contador=0;
        while(i<=5)
        {
            leer= JOptionPane.showInputDialog("Ingrese el Sueldo del Empleado " +i);
            sueldo= Double.parseDouble(leer);
            if(sueldo>300)
            contador= contador + 1;
            if(sueldo>mayor)
                mayor=sueldo;
            if(sueldo<menor)
                menor=sueldo;
            i=i+1;
        }
        JOptionPane.showMessageDialog(null, " El sueldo Mayor es de $: " + mayor +

```

```

        "\nEl sueldo Menor es de $: " + menor +
        "\n"+ contador + " Empleados tienen un sueldo mayor de $300");
    }
}

```

Ejemplo 11

- Nombre de la clase **CuadroDialog1**.

```

import javax.swing.JOptionPane;
public class CuadroDialog1 {
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        while (true){
            try{
                String nombre= JOptionPane.showInputDialog("Como te llamas?");
                String entrada= JOptionPane.showInputDialog("cuántos años tienes");
                int edad=Integer.parseInt(entrada);
                JOptionPane.showMessageDialog(null,"Hola, "+ nombre + ". El año que viene tendrás " +
                (edad+1) + "años","Resultado",JOptionPane.INFORMATION_MESSAGE);
            }
            catch(Exception e){
                System.out.println("Error en algun dato de entrada");
                JOptionPane.showMessageDialog(null,"Error en algun dato de
                entrada","Error",JOptionPane.ERROR_MESSAGE);
                String seleccion=(String) JOptionPane.showInputDialog(
                null,
                "Desea Salir",
                "Seleccione una opcion",
                JOptionPane.QUESTION_MESSAGE,
                null, // null para icono defecto
                new Object[] { "Si", "No" },
                "si");
                if (seleccion.equals("Si")){
                    System.exit(0);
                }
                break;
            }
        } //fin de catch
    } //fin de while
} //fin del metodo main
}

```

Tipos de iconos en JoptionPane

Crearemos una clase en la cual mostraremos las diferentes tipos de cajas para mostrar mensajes.

Crear una clase llamada **TiposIcono** y agregar el siguiente código.

```

import javax.swing.JOptionPane;

```

```

public class TiposIcono {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        //default title and icon
        JOptionPane.showMessageDialog(null,"Informacion","Mensaje De
Informacion",JOptionPane.INFORMATION_MESSAGE);
        JOptionPane.showMessageDialog(null,"Advertencia","Mensaje de
Advertencia",JOptionPane.WARNING_MESSAGE);
        JOptionPane.showMessageDialog(null,"Error","Mensaje de
Error",JOptionPane.ERROR_MESSAGE);
        JOptionPane.showMessageDialog(null,"Sin Icono","Mensaje de Texto
Plano",JOptionPane.PLAIN_MESSAGE);
    }
}

```

Resultados



Manejo de colecciones en Java

Ejemplo 12

Para este ejemplo crear una clase llamada “**EjemploCollection**”, en el cual se utilizara la interfaz Collection para el manejo de la información.

```

import java.util.ArrayList;
import java.util.Collection;

```

```

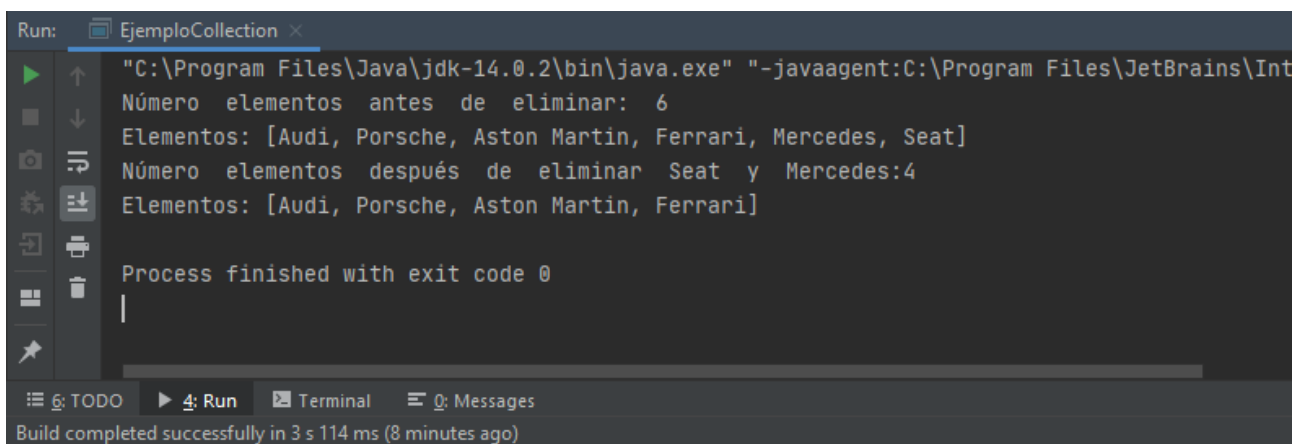
public class EjemploCollection {
    public static void main(String[] args) {
        Collection listaMarcasCoches = new ArrayList<String>(); //El tipo de listaMarcasCoches es
        Collection
        //Agregando información
        listaMarcasCoches.add("Audi");
        listaMarcasCoches.add("Porsche");
        listaMarcasCoches.add("Aston Martin");
        listaMarcasCoches.add("Ferrari");
        listaMarcasCoches.add("Mercedes");
        listaMarcasCoches.add("Seat");

        System.out.println("Número elementos antes de eliminar: " +
        listaMarcasCoches.size() ) ;//Obtener tamaño de Collection

        System.out.println ("Elementos: " +listaMarcasCoches.toString() ) ;
        listaMarcasCoches.remove ("Seat"); //Removiendo elemento por nombre
        listaMarcasCoches.remove ("Mercedes"); //Removiendo elemento por nombre
        System.out.println("Número elementos después de eliminar Seat y Mercedes:" +
        listaMarcasCoches.size() ) ;
        System.out.println("Elementos: " + listaMarcasCoches.toString() );
    }
}

```

Resultado



```

Run: EjemploCollection x
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\Int
Número elementos antes de eliminar: 6
Elementos: [Audi, Porsche, Aston Martin, Ferrari, Mercedes, Seat]
Número elementos después de eliminar Seat y Mercedes:4
Elementos: [Audi, Porsche, Aston Martin, Ferrari]

Process finished with exit code 0

```

Ejemplo13

Crear la clase llamada “**EjemploList**”, para esta clase se utiliza una interfaz de tipo ArrayList la cual posee ciertos métodos propios y que me proporcionan mayor libertad para el manejo de la información que está contenida en la Lista.

```

import java.util.*;
public class EjemploList{
    public static void main(String[] args) {
        ArrayList <String> listaPersona = new ArrayList();
        //Agregando elementos
        listaPersona.add("Marie Curie");
        listaPersona.add("Benjamin Franklin");
    }
}

```

```

listaPersona.add("Marco Antonio");
listaPersona.add(1, "Juan Murillo");
//Mostrar contenido de ArrayList
System.out.println("USO DE ARRAYLIST");
System.out.println("Contenido de listaPersona: " + listaPersona + "\n");
//Obteniendo posición de un elemento
int pos = listaPersona.indexOf("Benjamin Franklin");
System.out.println("El indice de Benjamin Franklin es: " + pos);
//Verificando si lista está vacía
boolean check = listaPersona.isEmpty();
System.out.println("Verificando si ArrayList está vacío: " + check);
//Obteniendo tamaño de lista
int size = listaPersona.size();
System.out.println("El tamaño de listaPersona es: " + size);
//Verificando si un elemento está incluido en la lista
boolean elemento = listaPersona.contains("Marco Antonio");
System.out.println("Verificando si 'Marco Antonio' está incluido en listaPersona: " +
elemento);
//Obteniendo elemento de una posición específica
String item = listaPersona.get(0);
System.out.println("El elemento con indice 0 es: " + item);
//Recuperando elementos del ArrayList
//Primera forma: Usando lazo for

System.out.println("\nRecuperando elementos del ArrayList con FOR");
for (int i = 0; i < listaPersona.size(); i++) {
    System.out.println("Indice: " + i + " - Elemento: " +
listaPersona.get(i));
}
//Segunda forma: Usando lazo foreach
System.out.println("\nRecuperando elementos del ArrayList con FOREACH");
for (String persona : listaPersona) {
    System.out.println("Elemento: " + persona);
}
//Tercera forma: Usando Iterator
//hasNext(): devuelve true si hay mas elementos
//next(): devuelve el siguiente elemento
System.out.println("\nRecuperando elementos del ArrayList con ITERATOR");
for (Iterator<String> it = listaPersona.iterator(); it.hasNext();)
{
    System.out.println("Elemento: " + it.next());
}

//Reemplazando un elemento
listaPersona.set(1, "Nikola Tesla");
System.out.println("\nArrayList después de reemplazo: " + listaPersona);
//Eliminando elemento en posición 0
listaPersona.remove(0);
System.out.println("\nArrayList después de eliminar elemento 0: " + listaPersona);
//Convirtiendo ArrayList en Array
String[] simpleArray = listaPersona.toArray(new String[listaPersona.size()]);
System.out.println("\nEl objeto Array creado es: " + Arrays.toString(simpleArray));
}

```



```
}
```

Resultado

```
USO DE ARRAYLIST
Contenido de listaPersona: [Marie Curie, Benjamin Franklin, Marco Antonio, Juan Murillo]

El indice de Benjamin Franklin es: 1
Verificando si ArrayList está vacío: false
El tamaño de listaPersona es: 4
Verificando si 'Marco Antonio' está incluido en listaPersona: true
El elemento con indice 0 es: Marie Curie

Recuperando elementos del ArrayList con FOR
Indice: 0 - Elemento: Marie Curie
Indice: 1 - Elemento: Benjamin Franklin
Indice: 2 - Elemento: Marco Antonio
Indice: 3 - Elemento: Juan Murillo

Recuperando elementos del ArrayList con FOREACH
Elemento: Marie Curie
Elemento: Benjamin Franklin
Elemento: Marco Antonio
Elemento: Juan Murillo

Recuperando elementos del ArrayList con ITERATOR
Elemento: Marie Curie
Elemento: Benjamin Franklin
Elemento: Marco Antonio
Elemento: Juan Murillo

ArrayList después de reemplazo: [Marie Curie, Nikola Tesla, Marco Antonio, Juan Murillo]

ArrayList después de eliminar elemento 0: [Nikola Tesla, Marco Antonio, Juan Murillo]

El objeto Array creado es: [Nikola Tesla, Marco Antonio, Juan Murillo]
```

Ejemplo 14

Crear la clase llamada “**UtilesEscolares**”, en este ejemplo se utiliza un Map que es un conjunto de valores, con el detalle de que cada uno de estos valores tiene un objeto extra asociado. A los primeros se los llama “claves” o “keys”, ya que son los que permiten acceder a los segundos.

```
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;

public class UtilesEscolares {
    public static void main(String[] args) {
        //Declarando HashMap
        HashMap<Integer, String> hmap = new HashMap<Integer, String>();
        //Agregando elementos
```

```

        hmap.put(12, "Crayolas");
hmap.put(2, "Lápices");
        hmap.put(7, "Borradores");
hmap.put(49, "Colores");
        hmap.put(3, "Boligrafos");
//Mostrando contenido usando Iterator
        System.out.println("Contenido de HashMap:");
Set set = hmap.entrySet();
        Iterator iterador = set.iterator();
        while(iterador.hasNext()) {
            Map.Entry mentry = (Map.Entry)iterador.next();
            System.out.println("Clave: " + mentry.getKey() + " - Valor: "
                + mentry.getValue());
        }
//Obtener valor en base a clave
        String var= hmap.get(2);
        System.out.println("\nValor asociado a clave 2: " + var);
//Remover elementos en base a clave
        hmap.remove(3);
        System.out.println("\nHashMap después de eliminar elemento con clave 3:");
Set set2 = hmap.entrySet();
        Iterator iterador2 = set2.iterator();
        while(iterador2.hasNext()) {
            Map.Entry mentry2 = (Map.Entry)iterador2.next();
            System.out.println("Clave: " + mentry2.getKey() + " - Valor: " + mentry2.getValue());
        }
    }
}

```

Resultado

```

Contenido de HashMap:
Clave: 49 - Valor: Colores
Clave: 2 - Valor: Lápices
Clave: 3 - Valor: Boligrafos
Clave: 7 - Valor: Borradores
Clave: 12 - Valor: Crayolas

Valor asociado a clave 2: Lápices

HashMap después de eliminar elemento con clave 3:
Clave: 49 - Valor: Colores
Clave: 2 - Valor: Lápices
Clave: 7 - Valor: Borradores
Clave: 12 - Valor: Crayolas

```

V. EJERCICIOS COMPLEMENTARIOS.

- Al ingresar la nota de un alumno, se desea saber si este aprobó o no una materia en el colegio. Para aprobar se necesita una nota mayor o igual a 7.0. Diseñe una aplicación en Java que al ingresar la nota muestre con un mensaje en el cual indique si el alumno aprobó o no. Además, si la nota está entre 6.50 y 6.99, tiene la posibilidad de realizar un examen de suficiencia para aprobar.

- En una tienda se realizan diferentes descuentos a sus clientes al momento de cancelar en caja. Cuando se dispone a cancelar tiene la oportunidad de sacar una bolita y dependiendo del color de la misma se le puede aplicar un determinado descuento. Si la bolita es roja, se le aplica un 10% de descuento sobre la compra; si la bola es verde, se le aplica un 5% de descuento y si la bolita es blanca, no tiene descuento y se le da las gracias por participar.
- Modificar el ejemplo 9 de tal forma que me permita leer el número de empleados de los cuales voy a ingresar el sueldo, además el programa no permitirá leer datos negativos, así que al ingresar un dato negativo me dará un error y me pedirá de nuevo el dato, si el dato sigue siendo negativo no me permitirá ingresar el sueldo de otro empleado hasta que se ingrese el valor positivo del sueldo.
- Desarrollar una aplicación que utilice un HashMap y en el cual se permita manejar la información de los alumnos de la materia de POO, para este caso se necesita tener una lista siendo la llave el “Carnet”, esta aplicación deberá de tener un menú en el cual se tenga como opciones las que se detallan a continuación.
 - Ingreso de estudiante.
 - Ver estudiante
 - Buscar Estudiante
 - Salir

VI. REFERENCIA BIBLIOGRAFICA.

- <http://download.netbeans.org/netbeans/6.1/final/>
- http://www.netbeans.org/index_es.html
- <http://es.wikipedia.org/wiki/NetBeans>
- **Aprendiendo Java 2 en 21 Días**
Lemay, Laura
- **Cómo Programar en Java**
Deitel, Harvey M.
- **Reloco.com.ar. (2016). Collections en Java. [online] Disponible en :**
<http://www.reloco.com.ar/prog/java/collections.html> [Accedido 28 Oct. 2016].