	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	68/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Guía práctica de estudio 05: Estructuras de selección




Elaborado por:

M.C. M. Angélica Nakayama C.
Ing. Jorge A. Solano Gálvez

Autorizado por:

M.C. Alejandro Velázquez Mena

	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	69/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Guía práctica de estudio 05: Estructuras de selección

Objetivo:

Implementar programas utilizando estructuras de selección en un lenguaje orientado a objetos.

Actividades:


- Conocer la sintaxis para declarar diversas estructuras de selección.
- Implementar el uso de estructuras de selección en un programa.

Introducción


Los lenguajes de programación tienen elementos básicos que se utilizan como bloques constructivos, así como reglas para que esos elementos se combinen. Estas reglas se denominan **sintaxis del lenguaje**. Solamente las instrucciones sintácticamente correctas pueden ser interpretadas por la computadora y los programas que contengan errores de sintaxis son rechazados por la máquina.

La **sintaxis** de un lenguaje de programación se define como el conjunto de reglas que deben seguirse al escribir el código fuente de los programas para considerarse como correctos para ese lenguaje de programación.

Las **estructuras de control** permiten modificar el flujo de ejecución de las instrucciones de un programa. Todas las estructuras de control tienen un único punto de entrada. Las estructuras de control se pueden clasificar en: secuenciales, transferencia de control e iterativas. Básicamente lo que varía entre las estructuras de control de los diferentes lenguajes es su sintaxis, cada lenguaje tiene una sintaxis propia para expresar la estructura.

	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	70/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

NOTA: En esta guía se tomará como caso de estudio el lenguaje de programación JAVA, sin embargo, queda a criterio del profesor el uso de éste u otro lenguaje orientado a objetos.

	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	71/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Estructuras de control

Las estructuras de programación o estructuras de control permiten **tomar decisiones** o **realizar un proceso repetidas veces**. En la mayoría de los lenguajes de programación, este tipo de estructuras son comunes en cuanto a concepto, aunque su sintaxis varía de un lenguaje a otro. La sintaxis de Java coincide prácticamente con la utilizada en C/C++, por lo que para un programador que maneja estos lenguajes no representa ninguna dificultad adicional.

Sentencias o expresiones

Una expresión es un conjunto de variables unidos por operadores. Son órdenes que se envían a la computadora para que realice una tarea determinada. Una sentencia es una expresión que acaba en punto y coma (;). Se permite incluir varias sentencias en una línea, aunque lo habitual es utilizar una línea para cada sentencia.

Ejemplo:

```
i = 0; j = 5; x = i + j; // Línea compuesta de tres sentencias
```


Estructuras de selección

Las estructuras de selección o bifurcaciones permiten ejecutar una de entre varias acciones en función del valor de una expresión lógica o relacional. Se tratan de estructuras muy importantes ya que son las encargadas de controlar el **flujo de ejecución** de un programa.

Java posee las estructuras de selección: **if-else**, **operador ternario** y **switch**.

IF / IF-ELSE

Esta estructura permite ejecutar un conjunto de sentencias en función del valor que tenga la expresión de comparación (se ejecuta si la expresión de comparación tiene valor **true**).

	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	72/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			


Las sentencias incluidas en el **else** se ejecutan en el caso de no cumplirse la expresión de comparación (**false**). Tiene la forma siguiente:

```
if (booleanExpression) {
    statements1;
} else {
    statements2;
}
```

```
/**
 * Se generan dos números aleatorios y se obtiene el valor absoluto
 * de la diferencia de los mismos.
 */
public class IfElse {
    public static void main (String []args){
        java.util.Random random = new java.util.Random();
        int one = random.nextInt();
        int two = random.nextInt();
        System.out.println("Los números generados son: ");
        System.out.println("one: " + one);
        System.out.println("two: " + two);
        if ((one-two) < 0)
            System.out.println("|" + one + " - " + two + "| = " + (one-two)*-1);
        else
            System.out.println("|" + one + " - " + two + "| = " + (one-two));
    }
}
```

Las llaves { } sirven para agrupar en un bloque las sentencias que se han de ejecutar, y no son necesarias si sólo hay una sentencia dentro del bloque **if** o **else**.

Si se desea introducir más de una expresión de comparación se usa **if / else if**. Si la primera condición no se cumple, se compara la segunda y así sucesivamente. En el caso de que no se cumpla ninguna de las comparaciones se ejecutan las sentencias correspondientes al **else**.

	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	73/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

```


if (booleanExpression1) {
    statements1;
} else if (booleanExpression2) {
    statements2;
} else if (booleanExpression3) {
    statements3;
} else {
    statements4;
}

```

```

/**
 * Programa que genera un número aleatorio de 0 a 6, los cuales
 * tienen una correspondencia con un día de la semana
 */
public class IfElseAnidado {
    public static void main (String [] args){
        // 0-> Domingo, 1-> Lunes, 2-> Martes, 3-> Miércoles
        // 4-> Jueves, 5-> Viernes, 6-> Sábado
        java.util.Random random = new java.util.Random();
        int one = random.nextInt();
        if (one < 0)
            one *= -1;
        one %= 7;
        System.out.println("One = " + one);
        if (one == 0 || one == 6)
            System.out.println("Día de descanso.");
        else if (one == 1)
            System.out.println("Inicio de semana.");
        else if (one == 5)
            System.out.println("Inicia el fin de semana.");
        else
            System.out.println("Entre semana.");
    }
}

```


	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	74/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

SWITCH

Se trata de una alternativa a la bifurcación **if/else if** cuando se compara la **misma expresión** con distintos valores.

Su forma general es la siguiente:

```
switch (expression) {
    case value1:
        statements1;
        break;
    case value2:
        statements2;
        break;
    case value3:
        statements3;
        break;
    case value4:
        statements4;
        break;
    case value5:
        statements5;
        break;
    case value6:
        statements6;
        break;
    [default:
        statements7;]
}
```

	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	75/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			


Las características más relevantes de **switch** son las siguientes:

- Cada sentencia **case** corresponde con un único valor de **expression**. No se pueden establecer rangos o condiciones, sino que se debe comparar con valores concretos de tipo **int** (incluyendo a los que se pueden convertir a **int** como **byte**, **char** y **short**) y **enumeraciones**.
- No puede haber dos etiquetas **case** con el mismo valor.
- Los valores que no están comprendidos en alguna sentencia case se pueden gestionar en **default**, que es **opcional**.
- En ausencia de **break**, cuando se ejecuta una sentencia case se ejecutan también todas las case que van a continuación, hasta que se llega a un **break** o hasta que se termina el **switch**.

```

/**
 * Programa que genera un número aleatorio de 0 a 6, los cuales
 * tienen una correspondencia con un día de la semana.
 */
public class Switch {
    public static void main (String [] args){
        // 0-> Domingo, 1-> Lunes, 2-> Martes, 3-> Miércoles
        // 4-> Jueves, 5-> Viernes, 6-> Sábado
        java.util.Random random = new java.util.Random();
        int one = random.nextInt();
        if (one < 0)
            one *= -1;
        one %= 7;
        System.out.println("One = " + one);
        switch (one) {
            case 0:
            case 6:
                System.out.println("Día de descanso.");
                break;
            case 1:
                System.out.println("Inicio de semana.");
                break;
            case 5:
                System.out.println("Inicia el fin de semana.");
                break;
            default:
                System.out.println("Entre semana.");
                break;
        }
    }
}

```


	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	76/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			


Operador ternario

El operador ternario es una estructura parecida a if-else, para cuando el bloque de código está constituido por una sola instrucción. La sintaxis del operador ternario es la siguiente:

condicion_logica ? expresion1: expresion2

El operador ternario evalúa la condición lógica, si se cumple (true) se ejecuta la instrucción que está a la derecha del ? (expresion1); si no se cumple (false) se ejecuta la instrucción que está a la derecha de los : (expresion2).

```
/**
 * Se genera un valor aleatorio en valor absoluto para después
 * comparar su valor en alguna opción del switch
 */
public class Ternario {
    public static void main (String [] args){
        // 0-> Domingo, 1-> Lunes, 2-> Martes, 3-> Miércoles
        // 4-> Jueves, 5-> Viernes, 6-> Sábado
        java.util.Random random = new java.util.Random();
        int one = random.nextInt();
        one = one < 0 ? one*-1: one;
        one %= 7;
        System.out.println("One = " + one);
        switch (one) {
            case 0:
            case 6:
                System.out.println("Día de descanso.");
                break;
            case 1:
                System.out.println("Inicio de semana.");
                break;
            case 5:
                System.out.println("Inicia el fin de semana.");
                break;
            default:
                System.out.println("Entre semana.");
                break;
        }
    }
}
```

	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	77/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

ENUMERADOR

Los enumeradores son listas que permiten almacenar valores constantes. Todos los valores que le pertenecen a un enumerador son estáticos y finales.

Para definir una enumeración en java se utiliza la palabra reservada **enum**. La sintaxis es la siguiente:


```
[modificadorAcceso] enum Nombre {
    VALOR1, VALOR2, VALOR3, ...
}
```

Debido a que los elementos de las enumeraciones son estáticos, el acceso a los mismos se realiza a través del nombre de la enumeración, seguido de punto y después el nombre del valor.

```
public enum Valores {
    VALOR1, VALOR2, VALOR3
}
```

```
Valores valor = Valores.VALOR1;
```

Los nombres de las enumeraciones siguen la misma convención que los nombres de las clases, es decir, ocupan notación Upper Camell Case. Así mismo, como los valores son constantes se escriben en mayúscula.

	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	78/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

```

/**
 * Se crea una enumeración para evaluar en el switch
 */
public class DiasSemana {
    public enum Dia {
        DOMINGO, LUNES, MARTES, MIERCOLES, JUEVES, VIERNES, SABADO
    }


    public void describirDia(Dia diaElegido) {
        switch (diaElegido) {
            case LUNES:
                System.out.println("Inicio de semana.");
                break;
            case VIERNES:
                System.out.println("Inicia fin de semana.");
                break;
            case SABADO:
            case DOMINGO:
                System.out.println("Día de descanso.");
                break;
            default:
                System.out.println("Entre semana.");
                break;
        }
    }
}

```

```

/**
 * Se prueban los valores de la enumeración
 */
public class PruebaDiasSemana {
    public static void main(String[] args) {
        DiasSemana dias = new DiasSemana();
        System.out.println("LUNES");
        dias.describirDia(DiasSemana.Dia.LUNES);
        System.out.println("MIÉRCOLES");
        dias.describirDia(DiasSemana.Dia.MIERCOLES);
        System.out.println("VIERNES");
        dias.describirDia(DiasSemana.Dia.VIERNES);
        System.out.println("SÁBADO");
        dias.describirDia(DiasSemana.Dia.SABADO);
    }
}

```

	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	79/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Clases de utilerías

En Java existen algunas clases que sirven para apoyar el desarrollo de aplicaciones, dichas clases tienen funcionalidades generales como por ejemplo cálculos matemáticos, fechas, etc. Algunas de las clases más útiles son: Math, Date y Calendar.

Math

Esta clase proporciona métodos para la realización de las **operaciones matemáticas** más habituales. Para utilizar sus métodos simplemente se utiliza el nombre de la clase Math seguida del operador punto y el nombre del método a utilizar.

Ejemplo:


```

Math.pow(5, 2);      //Eleva 5 a la potencia 2

Math.sqrt(25);       //Obtiene la raíz cuadrada de 25

Math.PI;             // Obtiene el valor aproximado de la constante PI

```

	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	80/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Date y Calendar

En el paquete `java.util` se encuentran dos clases para el tratamiento básico de fechas: **Date** y **Calendar**.

Un objeto **Date** representa una fecha y hora concretas con precisión de un milisegundo. Esta clase permite manipular una fecha y obtener información de la misma de una manera sencilla.

Para crear un objeto de la clase **Date** con la fecha y hora actual se utiliza:

```
Date fecha = new Date();
```


Usando el método `toString()` se obtiene la representación en forma de cadena de la fecha:

```
System.out.println(fecha.toString());
```

A partir de la versión 1.1 se incorporó una nueva clase llamada **Calendar** que amplía las posibilidades a la hora de trabajar con fechas, por tanto, es una clase que surgió para cubrir las carencias de la clase **Date** en el tratamiento de las fechas. Para crear un objeto de **Calendar** se usa la siguiente sintaxis:

```
Calendar calendario = Calendar.getInstance();
```

Utilizando el método `get()` se puede recuperar cada uno de los campos que componen la fecha, para ello este método acepta un número entero indicando el campo que se quiere obtener. La propia clase **Calendar** define una serie de **constantes** con los valores que corresponden a cada uno de los campos que componen una fecha y hora.

	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	81/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Ejemplo:

```
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;

public class Fechas {


    public static void main(String[] args) {
        Date fecha = new Date();
        System.out.println(fecha.toString());
        SimpleDateFormat formateador = new SimpleDateFormat("dd-MM-yyyy");
        System.out.println(formateador.format(fecha));

        Calendar calendario = Calendar.getInstance();
        String miFecha = "Hoy es día ";
        miFecha += calendario.get(Calendar.DAY_OF_MONTH) + " del mes ";
        miFecha += calendario.get(Calendar.MONTH)+ 1 + " de ";
        miFecha += calendario.get(Calendar.YEAR);
        System.out.println(miFecha);
    }
}
```

A partir de la versión **Java 8**, el manejo de las fechas y el tiempo ha cambiado en Java. Desde esta versión, se ha creado una nueva API para el manejo de fechas y tiempo en el paquete **java.time**, que resuelve distintos problemas que se presentaban con el manejo de fechas y tiempo en versiones anteriores.

Ejemplo:

```
LocalDate hoy = LocalDate.now();
System.out.println(hoy);
System.out.println(hoy.plusWeeks(1));
```

	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	82/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Bibliografía

Martín, Antonio

Programador Certificado Java 2.

Segunda Edición.

México

Alfaomega Grupo Editor, 2008

Sierra Katy, Bates Bert

SCJP Sun Certified Programmer for Java 6 Study Guide

Mc Graw Hill

Dean John, Dean Raymond.

Introducción a la programación con Java

Primera Edición.

México

Mc Graw Hill, 2009