FACULTAD DE INGENIERÍA

FORMATO
PRÁCTICAS DE LABORATORIO

UNIVERSIDAD AUTÓNOMA DE CAMPECHE

NOMBRE DEL PROFESOR: Ing. Héctor Manuel Quej Cosgaya
NOMBRE DE LA PRÁCTICA: Estructuras de iteración
PRÁCTICA NÚM. [6]

LABORATORIO:	Centro de Ingeniería Computacional
MATERIA:	Lenguaje de Programación I
UNIDAD:	Subcompetencia III
TIEMPO:	2 horas

OBJETIVO:

Comprender el funcionamiento, características, ventajas y desventajas de las diferentes estructuras de iteración con los que cuenta el lenguaje de programación Java.

MARCO TEÓRICO:

Los programas de computadora, a menudo, deben repetir ciertas tareas de manera repetitiva. Para que el programador no tenga que escribir una y otra vez esas tareas, los lenguajes de programación ofrecen elementos llamados estructuras de iteración, que se encargan de repetir la ejecución de las instrucciones que contienen una y otra vez, hasta que se cumpla determinada condición. Dado que son elementos básicos de cualquier programa, es fundamental aprender a utilizarlos, así como conocer sus diferencias para saber cual utilizar en que situación. En la práctica, se procede a mostrar los tres diferentes tipos de ciclos que el lenguaje de programación Java posee, así como las estructuras de ramificación.

LISTA DE MATERIALES:

- Java SDK
- Entorno de Desarrollo Integrado NetBeans 7.0 o superior

EQUIPO DE LABORATORIO:

- Computadora Personal

DESARROLLO DE LA PRÁCTICA:

Primera parte: **Ciclo While**

1. Abre el Entorno de Desarrollo Integrado (IDE) **NetBeans**. Normalmente, hay un acceso directo en el Escritorio para él.
2. **Abre** el **Proyecto** que creaste en la **Práctica #5**.
3. Crea una nueva **Clase Java** dentro de tu proyecto recién abierto. Nómbrala "**Practica6**".

Si tienes dudas acerca de como utilizar el IDE NetBeans, consulta la **Práctica #5** de nuevo.

4. **Borra** todos los comentarios añadidos por el IDE, de manera que la declaración de la clase "public class Practica6…" sea la línea número 1.
5. Añade el siguiente fragmento de código dentro de la clase "Practica6":

```
public static void main(String[] args) {
    // Primera parte: Ciclo While
    java.util.Scanner leer = new java.util.Scanner(System.in);
    String respuesta = "si";
    System.out.println("Entrando al while");
    while (respuesta.equals("si")) {
        System.out.println("Introduce un número:");
        int numero = leer.nextInt();
        System.out.println("Número " + numero + " leído");
        System.out.println("¿Deseas continuar? (si/no)");
        respuesta = leer.next();
    }
    System.out.println("Estamos fuera del while");
```

6. Añade ***breakpoints*** en las líneas **7, 10** y **14** haciendo clic en la pequeña barra de la izquierda que lleva la cuenta las líneas. Tu código debe lucir como el siguiente:

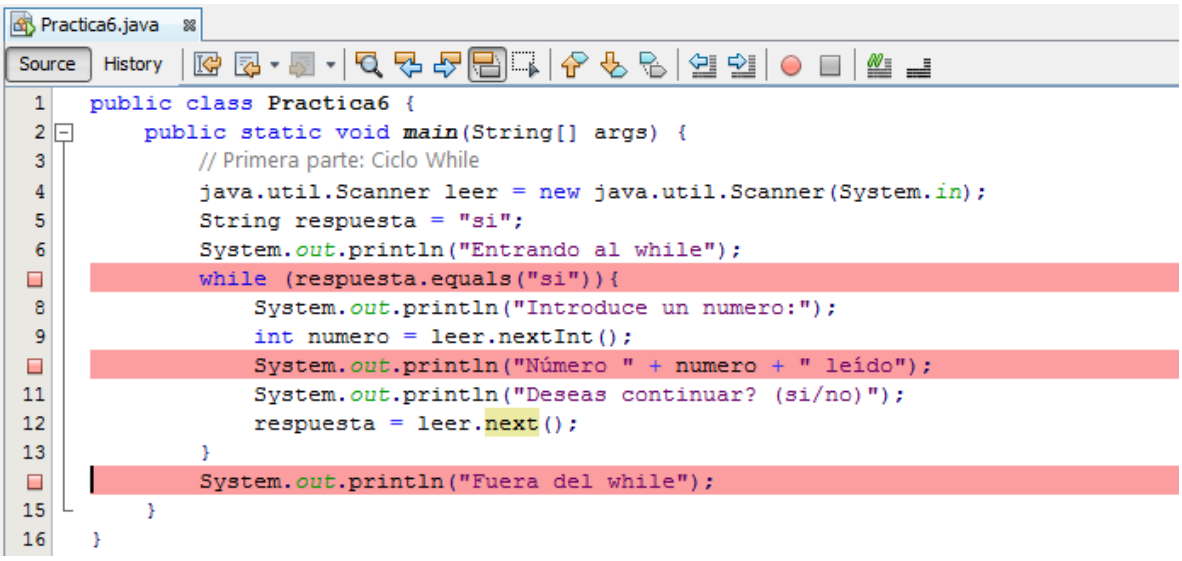

Fig. 1 – Breakpoints para el ciclo While

7. Ejecuta el programa en modo **Depuración**. Introduce un número cuando el programa te lo solicite. Escribe "si" para continuar dentro del ciclo **While**. Escribe cualquier otra cosa para salir.

8. En la declaración de respuesta, cambia su valor inicial por “no”. Ejecuta de nuevo el programa en modo **Depuración** y observa lo que sucede.
9. **Borra** todos los ***breakpoints*** que creaste antes de pasar a la siguiente sección.

Segunda parte: **Ciclo Do – While.**

10. Añade el siguiente fragmento de código después de la línea 14.

```
do {
   System.out.println(“¡Yo si me ejecuto!”);
   System.out.println(“¿Deseas continuar? (si/no)”);
   respuesta = leer.next();
} while (respuesta.equals(“si”));
System.out.println(“Fuera del do – while”);
```

11. Añade un ***breakpoint*** en las líneas **16** y **19**. Tu código debe lucir como el siguiente:

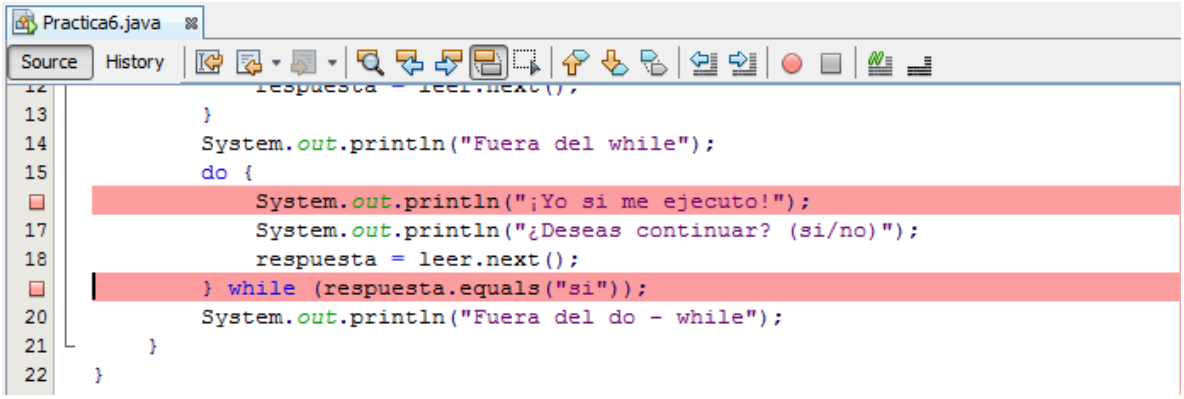

Fig. 2 – Breakpoints para el Ciclo Do - While

12. Ejecuta el programa de nuevo en modo **Depuración**. Observa que el ciclo **While** es completamente ignorado, pero el ciclo **Do – While** se ejecuta al menos una vez. Escribe “no” para salir del ciclo.
13. Borra todos los ***breakpoints*** que creaste hasta el momento antes de pasar a la siguiente sección haciendo clic sobre ellos.

Tercera parte: **Ciclo For / For Each**

14. Añade el siguiente fragmento de código después de la línea **20**:

```
// Tercera parte: Ciclo For / For Each
int[] cuadrados = new int[10];
System.out.println(“Entrando al For”);
for (int i = 0; i < cuadrados.length; i++){
   System.out.println(“i es ahora: “ + i);
   cuadrados[i] = (i * i);
```

```
}
System.out.println("Fuera del For – Entrando al For Each");
for(int n : cuadrados) {
   System.out.println(n);
}
System.out.println("Fuera del For Each");
```

15. Añade ***breakpoints*** en las **25** y **30**. Tu código debe lucir como el siguiente:

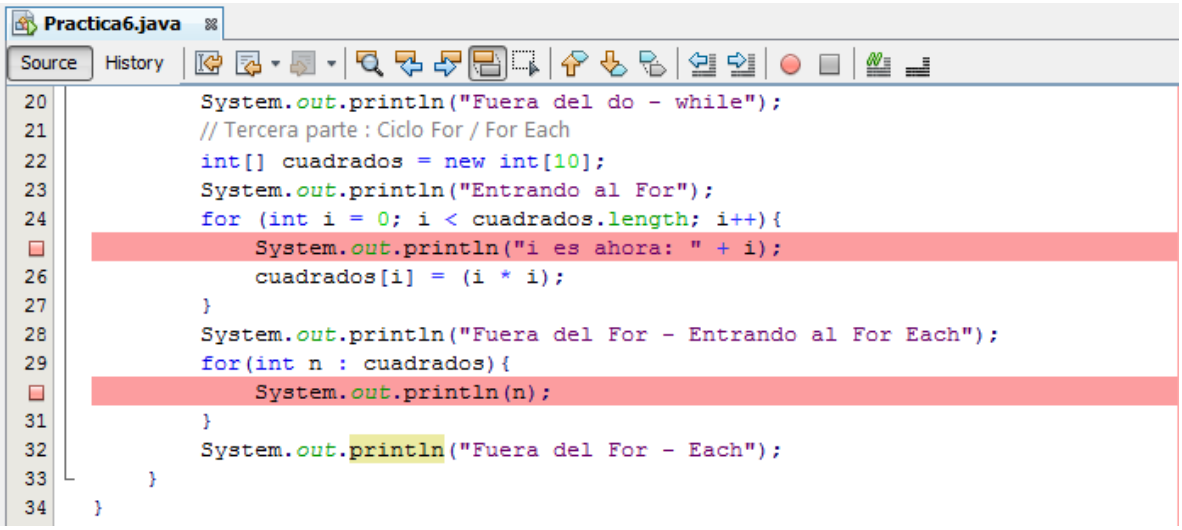

Fig. 3 – Los Ciclos For y For Each

16. Ejecuta el programa de nuevo en modo **Depuración**. Observa el camino que sigue el flujo del programa.
17. Borra todos los ***breakpoints*** que añadiste en esta sección antes de pasar a la siguiente parte.

Cuarta parte: **Sentencias de ramificación**

18. Añade el siguiente fragmento de código después de la línea **32**:

```
// Cuarta parte: Sentencias de ramificación
String[] animales = {"perro", "gato", null, "canario", null};
for (String p : animales) {
   System.out.println("Principio del For – Each");
   if (p == null) continue;
   System.out.println(p + " tiene " + p.length() + " letras");
}
```

19. Añade ***breakpoints*** en las líneas **36, 37,** y **38**. Tu código debe lucir como el siguiente:

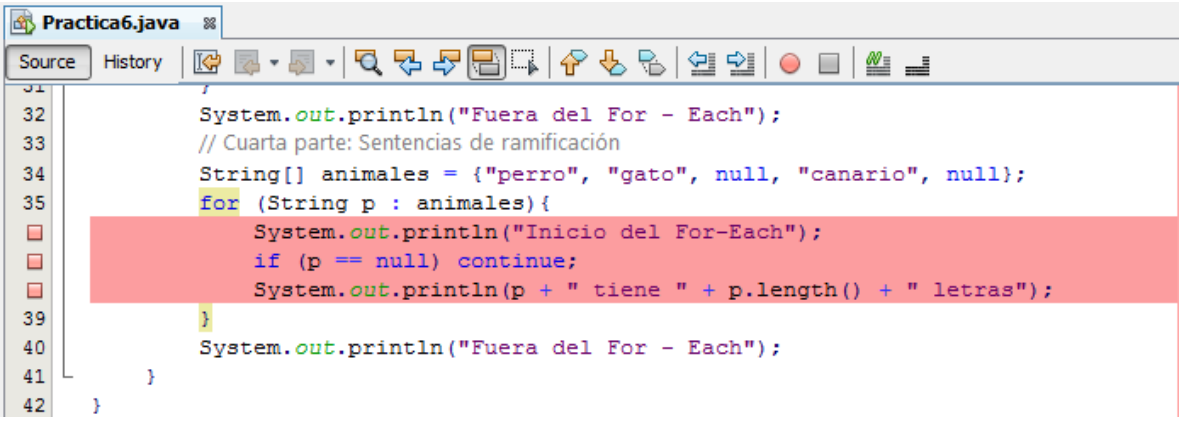

Fig. 4 – La sentencia de ramificación continue

20. Ejecuta el programa en modo **Depuración**. Observa el camino que sigue el flujo del programa.
21. Cambia la sentencia de ramificación `continue` por la sentencia de ramificación `break`.
22. Ejecuta el programa en modo **Depuración**. Observa el cambio en el comportamiento del programa.

Fin de la práctica.

RETROALIMENTACIÓN:

- Reescribe el programa que hiciste durante la práctica 2, ahora utilizando ciclos de iteración y/o decisión
- Escribe un programa que permita leer dos matrices de 3 x 3, y que calcule su producto.

RECOMENDACIONES ADICIONALES:

- Investiga en qué condiciones es mejor utilizar cada uno de las estructuras de control (iteración y decisión) (por ejemplo, cuando es mejor usar un switch que un if-else)
- Lee el capítulo 4 del Dean (Estructuras de control)

BIBLIOGRAFÍA:

- Dean, J. S., & Dean, R. H. (2009). Introducción a la programación con Java. México: Mc Graw Hill.
- Roberts, Simon; Heller Philip y Ernest, Michael (1999). The Complete Java 2 Certification Study Guide. Alameda, California: SYBEX.
- Apuntes del profesor.

FACULTAD DE INGENIERÍA

FORMATO
PRÁCTICAS DE LABORATORIO

UNIVERSIDAD AUTÓNOMA DE CAMPECHE