WPA2 & Ettercap

Jesús Rodríguez Heras
Juan Pedro Rodríguez Gracia

6 de noviembre de 2017

Resumen

Definición del protocolo WPA2 con la explicación del ataque KRACK y ejemplo de ataque con Ettercap.

Índice

1. Problemas resueltos

1.1. `Circulo.java`

Escriba un programa en java para calcular el volumen de un cono. Declare una constante que guarde el valor de π. Suponga un cono de 14,2 cm de diámetro en la base y de 20 cm de altura. Guárdelo en un fichero llamado `Circulo.java`.

```
1
2  /**
3   * @author Jesús Rodríguez Heras
4   * @version 1.0
5   */
6  public class Circulo {
7
8      static final double PI = 3.141592;
9
10     /**
11      * @param diametroBase = Diámetro de la base
12      * @param altura = Altura del cono
13      */
14     public static void volumen(double diametroBase, double altura) {
15         double v = (altura * PI * Math.pow(diametroBase / 2, 2)) / 3;
16         System.out.println("El volumen del cono es: " + v);
17         //El resultado estará en centímetros cuadrados
18     }
19
20     public static void main(String[] args) {
21         double diametroBase = 14.2;
22         double altura = 20;
23         //Ambas medidas en centímetros
24         volumen(diametroBase, altura);
25     }
26 }
```

1.2. `NewtonRaphson.java`

Escriba un programa en java para encontrar el cero de una función $f(x)$ mediante el método de *Newton-Raphson*. Este método iterativo construye una sucesión x_0, x_1, x_2... de aproximaciones a la solución utilizando la siguiente ecuación:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

La aproximación inicial será introducida por teclado, junto con el número de iteraciones que permitirán obtener la aproximación a la raíz de la función $f(x)$. El programa irá imprimiendo en pantalla las sucesivas aproximaciones que va calculando. Aplique su programa a las funciones siguientes:

- $f(x) = cos(x) - x^3$ **en [0, 1]**

- $f(x) = x^2 - 5$ **en [2, 3]**

Guarde su programa en un fichero llamado `NewtonRaphson.java`.

```
1
2  /**
3   * @author Jesús Rodríguez Heras
4   * @version 1.0
5   */
6  import java.util.Scanner;
7
8  public class NewtonRaphson {
9
10     public static Scanner teclado = new Scanner(System.in);
11
12     /**
13      * @param iteraciones Número de repeticiones
14      */
15     public static void funcion1(int iteraciones) {
16         double aproximacion, fn, fn_p;
17         int i = 1;
18
19         System.out.println("Introduce una aproximación para la función:");
20         aproximacion = teclado.nextDouble();
21
22         while (i <= iteraciones) {
23             fn = Math.cos(aproximacion) - (Math.pow(aproximacion, 3));
24             fn_p = -Math.sin(aproximacion) - (3 * Math.pow(aproximacion, 2));
25             aproximacion = aproximacion - (fn / fn_p);
26
27             System.out.println("Iteración " + i + ": " + aproximacion + ".");
28             i++;
29         }
30     }
31
32     /**
33      * @param iteraciones Número de repeticiones
34      */
35     public static void funcion2(int iteraciones) {
36         double aproximacion, fn, fn_p;
37         int i = 1;
38
39         System.out.println("Introduce una aproximación para la función:");
40         aproximacion = teclado.nextDouble();
41
42         while (i <= iteraciones) {
43             fn = Math.pow(aproximacion, 2) - 5;
44             fn_p = 2 * aproximacion;
45             aproximacion = aproximacion - (fn / fn_p);
46
47             System.out.println("Iteración " + i + ": " + aproximacion + ".");
```

```
48              i++;
49          }
50      }
51
52      public static void main(String[] args) {
53          int funcion, iteraciones;
54
55          do {
56              System.out.println("Elige una función:");
57              System.out.println("1.) f(x) = cos(x) - x^3 en [0, 1]");
58              System.out.println("2.) f(x) = x^2 - 5 en el [2, 3]");
59              funcion = teclado.nextInt();
60          } while (funcion != 1 && funcion != 2);
61
62          System.out.println("Introduce el número de iteraciones:");
63          iteraciones = teclado.nextInt();
64
65          switch (funcion) {
66              case 1:
67                  funcion1(iteraciones);
68                  break;
69              case 2:
70                  funcion2(iteraciones);
71                  break;
72          }
73      }
74  }
```

1.3. `intDefinidaMonteCarlo.java`

La integral definida en [0 - 1] de una función real de variable real $f(x)$ puede calcularse mediante un método de Monte Carlo (probabilístico) inscribiendo la curva de la función en un cuadrado de lado igual a la unidad. Para aproximar el valor de la integral, se generan puntos aleatorios en el marco determinado por el cuadrado, y se cuentan únicamente aquellos puntos que están situados bajo la curva. La razón entre el número de puntos bajo la curva y el número total de puntos es una aproximación al valor buscado que naturalmente, conforme mayor es el número de puntos, mejora la aproximación. Escriba un programa java que permita realizar tal cálculo, leyendo desde teclado el número de puntos con el cuál genera la aproximación para las funciones siguientes:

- $f(x) = sin(x)$
- $f(x) = x$

Guarde el programa en `intDefinidaMonteCarlo.java`.

```
1
2   /**
3    * @author Jesús Rodríguez Heras
4    * @version 1.0
```

```
 */
import java.util.Scanner;

public class intDefinidaMonteCarlo {

    public static Scanner teclado = new Scanner(System.in);

    /**
     * @param puntos = Número de puntos totales
     */
    public static void funcion1(int puntos) {
        double puntosdentro = 0;
        double x, y, buscado;

        for (int i = 0; i < puntos; i++) {
            x = Math.random();
            y = Math.random();
            if (Math.sin(x) >= y) {
                puntosdentro++;
            }
        }
        buscado = puntosdentro / puntos;
        System.out.println("La integral definida en [0, 1] es: " + buscado);
    }

    /**
     * @param puntos = Número de puntos totales
     */
    public static void funcion2(int puntos) {
        double puntosdentro = 0;
        double x, y, buscado;

        for (int i = 0; i < puntos; i++) {
            x = Math.random();
            y = Math.random();
            if (x >= y) {
                puntosdentro++;
            }
        }
        buscado = puntosdentro / puntos;
        System.out.println("La integral definida en [0, 1] es: " + buscado);
    }

    public static void main(String[] args) {
        int funcion;

        do {
            System.out.print("Seleccione la función a la que le desea");
            System.out.println(" calcular la integral en el intervalo [0, 1]");
            System.out.println("1.) f(x)=sin(x)");
```

```
55              System.out.println("2.) f(x)=x");
56              funcion = teclado.nextInt();
57          } while (funcion != 1 && funcion != 2);
58
59          System.out.println("Introduce el número de puntos:");
60          int puntos = teclado.nextInt();
61
62          switch (funcion) {
63              case 1:
64                  funcion1(puntos);
65                  break;
66              case 2:
67                  funcion2(puntos);
68                  break;
69          }
70      }
71  }
```

1.4. `Cesar.java`

El cifrado de César es una técnica elemental de ocultamiento de la información que matemáticamente se describe de forma simple utilizando la siguiente ecuación

$$E(x) = x + n \ mod \ 27$$

donde x es la letra que queremos cifrar (representada por su código ASCII o por cualquier otra ordenación válida) y n es un número que se suma a ese código. Escriba un programa `Cesar.java` que lea el valor de n, una cadena de texto cualquiera, y muestre en pantalla su representación cifrada. Escriba otro programa llamado `desCesar.java` que efectúe el descifrado de acuerdo a la siguiente ecuación:

$$D(k) = k - n \ mod \ 27$$

1.4.1. `Cesar.java`

```
1
2   /**
3    * @author Jesús Rodríguez Heras
4    * @version 1.0
5    */
6   import java.util.Scanner;
7
8   public class Cesar {
9
10      /**
11       * @param cadena = Cadena a cifrar
12       * @param desplazamiento = Desplazamiento seleccionado por el usuario
13       */
14      public static void cifrar(String cadena, int desplazamiento) {
```

```
15          StringBuilder cifrado = new StringBuilder();
16          int valorASCII;
17
18          for (int i = 0; i < cadena.length(); i++) {
19              valorASCII = (int) (cadena.charAt(i));
20              valorASCII = valorASCII + (desplazamiento % 27);
21              cifrado.append((char) (valorASCII));
22          }
23
24          System.out.println("La␣cadena␣cifrada␣es:\n" + cifrado.toString());
25      }
26
27      public static void main(String[] args) {
28          int desplazamiento;
29
30          Scanner teclado = new Scanner(System.in);
31          System.out.println("Introduzca␣la␣cadena␣a␣codificar:");
32          String cadena = teclado.nextLine();
33
34          do {
35              System.out.println("Introudzca␣el␣desplazamiento␣(entre␣0␣y␣27):");
36              desplazamiento = teclado.nextInt();
37          } while (desplazamiento < 0 && desplazamiento > 27);
38
39          cifrar(cadena, desplazamiento);
40      }
41  }
```

1.4.2. desCesar.java

```
1
2   /**
3    * @author Jesús Rodríguez Heras
4    * @version 1.0
5    */
6   import java.util.Scanner;
7
8   public class desCesar {
9
10      /**
11       * @param cadena = Cadena a descifrar
12       * @param desplazamiento = Desplazamiento seleccionado por el usuario
13       */
14      public static void descifrar(String cadena, int desplazamiento) {
15          StringBuilder descifrado = new StringBuilder();
16          int ASCII;
17
18          for (int i = 0; i < cadena.length(); i++) {
19              ASCII = (int) (cadena.charAt(i));
20              ASCII = ASCII - (desplazamiento % 27);
```

```
            descifrado.append((char) (ASCII));
        }

        System.out.println("La cadena es:\n" + descifrado.toString());
    }

    public static void main(String[] args) {
        int desplazamiento;

        Scanner teclado = new Scanner(System.in);
        System.out.println("Introduzca la cadena a descodificar:");
        String cadena = teclado.nextLine();

        do {
            System.out.println("Introudzca el desplazamiento (entre 0 y 27):");
            desplazamiento = teclado.nextInt();
        } while (desplazamiento < 0 && desplazamiento > 27);

        descifrar(cadena, desplazamiento);
    }
}
```

1.5. aleatorios.java

Una tarea que será de utilidad durante el curso es la generación de números aleatorios. En Java, esto puede lograrse de dos maneras diferentes. Comencemos ahora por la primera; escriba un programa llamado `aleatorios.java` que use el método `random()` de la clase `Math` para generar una secuencia de números aleatorios. La longitud de la secuencia será fijada mediante un argumento leído por el programa desde la línea de comandos de una ventana de terminal.

```

/**
 * @author Jesús Rodríguez Heras
 * @version 1.0
 */
public class aleatorios {

    /**
     * @param args = Argumento leido por la línea de comandos
     */
    public static void numerosAleatorios(String[] args) {
        int numeros = Integer.parseInt(args[0]), i = 1;
        while (i <= numeros) {
            System.out.println("Número " + i + ": " + Math.random());
            //Solo imprimirá valores entre 0 y 1 debido a que el método
            //Math.random() solo devuelve valores aleatorios entre 0 y 1
            i++;
        }
    }

```

```
21      public static void main(String[] args) {
22          numerosAleatorios(args);
23      }
24  }
```

2. Referencias bibliográficas