Ejercicios de la UD05

1. Seguridad

1. Ejercicio 1:

   Una aplicación antimalware es a la vez:

   1. Seguridad lógica y activa.
   2. Seguridad física y activa.
   3. Seguridad lógica y pasiva.

2. Ejercicio 2:

   Cataloga los siguientes riesgos y amenazas segun si son físicos o lógicos.

Amenaza o Riesgo	**Tipo (F/L)**
SPAM	
Fuga en una tubería que pasa por encima del CPD	
Una obra cercana ha cortado el suministro eléctrico de toda la manzana	
Fuertes lluvias provocan inundaciones	
Debido a la llegada del verano se incrementan las temperaturas	
Phising	
Una persona consigue una tarjeta de identificación y consigue acceso al CPD sin estar autorizado	
Una persona de manera involuntaria provoca un incendio en la oficina	
Un usuario descarga un ejecutable y desactiva el software antimalware para poder ejecutarlo	

3. Ejercicio 3:

   Indica cuáles de las siguientes afirmaciones son verdaderas:

   1. Es imposible conseguir a la vez un sistema seguro, fácil de usar y funcional.
   2. La auditoria de seguridad informática no requiere de un permiso previo si notificamos los problemas que detectemos.
   3. Gracias al no repudio podemos estar seguros que la información que envió el emisor es la que nos ha llegado.
   4. La disponibilidad de la información garantiza que sea accesible desde cualquier lugar y en cualquier momento.
   5. Es posible conseguir la seguridad plena en un sistema informático.

4. Ejercicio 4:

   Rellena el siguiente texto con la información aprendida durante este punto.

   La seguridad _____ es la encargada de gestionar la seguridad los dispositivos físicos y controlar los elementos _____ como por ejemplo la temperatura y humedad relativa. Por contra la seguridad _____ se ocupa de la seguridad de los elementos de software y la _____ del usuario, así como de controlar el software malicioso, también llamado _____.

   (a) ambientales, (b) física, (c) información, (d) lógica, (e) malware

5. Ejercicio 5:

   Relaciona las diferentes vulnerabilidades con los sistemas de protección recomendados:

Vulnerabilidad	**Protección**
Malware.	Soluciones antivirus i antimalware.
Inundaciones.	Soluciones técnicas de construcción.
Terremotos.	CPD estanco.
Ransomware	Copias de seguridad
Ataque de ingeniería social.	Formación y entrenamiento a los usuarios

6. Ejercicio 6:

   Indica cuáles de las siguientes afirmaciones son verdaderas:

   1. Una vulnerabilidad 0-day es poco conocida y todavia no dispone de un parche que resuelva el problema.
   2. Garantizar la seguridad 100% es posible, solo necesitas los recursos económicos suficientes.
   3. El mejor lugar para colocar un servidor es uno que sea accesible a todo el personal de la empresa en cualquier momento.
   4. Aunque el CPD tenga muchas medidas de seguridad, es recomendable disponer de un segundo CPD o CPD de respaldo para asegurar la continuación del negocio en caso de desastre.
   5. Es mejor realizar las actualizaciones del sistema solo cuando detectemos problemas para evitar que estas desestabilicen nuestro sistema.

7. ¿Cómo se llama al par identificador-contraseña de un sistema de seguridad?
   a) Perfil.
   b) Credencial.
   e) Autorización.
   d) Rol.

8. ¿Cuál de las siguientes características físicas no puede ser un control de acceso biométrico?
   a) La voz.
   b) La huella dactilar.
   c) El reconocimiento facial.
   d) La altura.

9. Algunos algoritmos criptográficos con el paso del tiempo han dejado de ser seguros. Indica las razones principales.

10. Define en qué consiste el control de acceso basado en credenciales.

2. Excepciones

1. Vamos a programar la gestión de excepciones para leer 4 números enteros por teclado, nuestro profesor plantea tres posibles soluciones:

   a) Un solo bloque try que contiene la lectura de todos los números.

   b) Un bloque try..catch para cada lectura

   c) Escribir un método para realizar la lectura del número entero. Dentro del método será donde se controle la excepción.

   Explica e implementa la solución que más te guste, justifica en un comentario porque las otras dos soluciones no te gustan y cuales son sus principales inconvenientes.

2. Tratamiento de excepciones para leer dos números de tipo int, un String y dos números de tipo double y mostrar los valores leídos por pantalla. Se realizará un método para leer un número int y otro para leer un double. En estos métodos se realiza el control de excepciones en la lectura por teclado.

3. En este caso se proporciona un código Java y se pide tratar las excepciones que se pueden producir.

```java
public class Excepciones {

    static Scanner sc = new Scanner(System.in);

    public static void main(String[] args) {

        double n;
        int posicion;
        String cadena ;
        double[] valores = {9.83, 4.5, -3.06, 0.06, 2.52, -11.3, 7.60, 3.00, -30.4, 105.2};

        System.out.println("Contenido del array antes de modificar:");
        for (int i = 0; i < valores.length; i++) {
            System.out.printf("%.2f ", valores[i]);
        }

        System.out.print("\n\nIntroduce la posición del array a modificar: ");
        cadena = sc.nextLine();
        posicion = Integer.parseInt(cadena);

        System.out.print("\nIntroduce el nuevo valor de la posición " + posicion + ": ");
        n = sc.nextDouble();

        valores[posicion] = n;

        System.out.println("\nPosición a modificar " + posicion);
        System.out.println("Nuevo valor: " + n);
        System.out.println("Contenido del array modificado:");
        for (int i = 0; i < valores.length; i++) {
```

```
            System.out.printf("%.2f ", valores[i]);
        }

    }
}
```

4. Escribir un programa que divida dos números que se reciben en main en `args[0]` y `args[1]`.

   Ejemplo:

```
$ java dividir 10 5
10/5 es igual a 2
```

   Donde 10 y 5 son `args[0]` y `args[1]` respectivamente, es decir los parámetros con que llamamos al programa dividir.

5. Justifica por qué se produce error en el siguiente fragmento de código

```
try {
    System.out.println("Introduce edad: ");
    int edad = tec.nextInt();
    if (edad >= 18) {
        System.out.println("Mayor edad");
    } else {
        System.out.println("Menor edad");
    }
    System.out.println("Introduce nif");
    String nif = tec.next();
    int numero = Integer.parseInt(nif.substring(0, nif.length() -
1));
    char letra = nif.charAt(nif.length() - 1);
    System.out.println("Numero: " + numero);
    System.out.println("Letra: " + letra);
} catch (Exception e){
    System.out.println("Debías introducir un número");
} catch (NumberFormatException e) {
    System.out.println("El nif es incorrecto");
}
```

6. Indica qué se mostrará por pantalla cuando se ejecute esta clase y por qué:

```
public class Uno {
      private static int metodo()  {
            int valor=0;
            try  {
                  valor = valor + 1;
                  valor = valor + Integer.parseInt("42") ;
                  valor = valor + 1;
                  System.out.println("Valor al final del try: " +
valor);
            } catch(NumberFormatException e)  {
                  valor = valor + Integer.parseInt ("42");
```

```
                  System.out.println("Valor al final del catch: " +
valor) ;
            }
            finally  {
                  valor = valor + 1;
                  System.out.println("Valor al final de finally: " +
valor) ;
            }
            valor = valor + 1;
            System.out.println ("Valor antes del return: " + valor)
;
            return valor;
      }

      public static void main(String[] args)  {
            try {
                  System.out.println (metodo());
            }  catch (Exception e)  {
                  System.err.println("Excepcion en metodo()") ;
                  e.printStackTrace();
            }
      }
}
```

7. Indica qué se mostrará por pantalla cuando se ejecute esta clase y por qué:

```
public class Dos {
      private static int metodo()  {
            int valor=0;
            try   {
                  valor = valor+1;
                  valor = valor + Integer.parseInt("W");
                  valor = valor + 1;
                  System.out.println("Valor al final del try: " +
valor);
            } catch(NumberFormatException e) {
                  valor = valor + Integer.parseInt("42");
                  System.out.println("Valor al final del catch: " +
valor) ;
            } finally {
                  valor = valor + 1;
                  System.out.println("Valor al final de finally: " +
valor) ;
            }
            valor = valor + 1;
            System.out.println ("Valor antes del return: " + valor)
;
            return valor ;
      }

      public static void main (String[] args)  {
            try  {
                System .out.println(metodo());
            }  catch (Exception e) {
                  System.err.println("Excepcion en metodo() ");
```

```
                    e.printStackTrace();
              }
        }
}
```

8. Indica qué se mostrará por pantalla cuando se ejecute esta clase y por qué:

```
public class Tres {
      private static int metodo()  {
            int valor = 0;
            try {
                  valor = valor +1;
                  valor = valor + Integer.parseInt("W");
                  valor = valor + 1;
                  System.out.println("Valor al final del try : " +
valor);
            } catch (NumberFormatException e) {
                  valor = valor + Integer.parseInt("W");
                  System.out.println("Valor al final del catch : " +
valor);
            } finally {
                  valor = valor + 1;
                  System.out.println("Valor al final de finally: " +
valor);
            }
            valor = valor + 1;
            System.out.println ("Valor antes del return: " + valor);
            return valor ;
      }

      public static void main (String[ ] args)
      {
            try {
                  System.out.println(metodo ());
            } catch (Exception e) {
                  System.err.println("Excepcion en metodo()") ;
                  e.printStackTrace();
            }
      }
}
```

9. Indica qué se mostrará por pantalla cuando se ejecute esta clase y por qué:

```
import java.io.*;

public class Cuatro
{
      private static int metodo()  {
            int valor = 0;
            try {
                  valor = valor+1;
                  valor = valor + Integer.parseInt("W");
                  valor = valor + 1;
```

```
                  System.out.println("Valor al final del try : " +
valor) ;
                  throw new IOException();
            } catch (IOException e)  {
                  valor = valor + Integer.parseInt("42");
                  System.out.println("Valor al final del catch : " +
valor);
            } finally {
                  valor = valor + 1;
                  System.out.println("Valor al final de finally: " +
valor);
            }
            valor = valor + 1;
            System.out.println ("Valor antes del return: " + valor)
;
            return valor ;
      }

      public static void main(String[] args)  {
            try {
                  System.out.println(metodo());
            } catch (Exception e) {
                  System.err.println("Excepcion en metodo()");
                  e.printStackTrace();
            }
      }
}
```

10. Indica qué se mostrará por pantalla cuando se ejecute esta clase:

    1. Si se ejecuta con java Cinco casa
    2. Si se ejecuta con java Cinco 0
    3. Si se ejecuta con java Cinco 7

```
public class Cinco {
   public static void main(String args[])  {
      try  {
            int a = Integer.parseInt(args[0]);
            System.out.println("a = " + a);
            int b=42/a;
            String c = "hola";
            char d = c.charAt(50);
      }  catch (ArithmeticException e) {
         System.out.println("div por 0: " + e);
      }  catch (IndexOutOfBoundsException e) {
         System.out.println("Índice del String fuera de límites: " +
e);
      }  finally {
          System.out.println("Ejecución de finally");
      }
   }
}
```

11. Indica cuál será la salida del siguiente programa y por qué

```java
public class Seis {
   public static void procA()  {
       try {
           System.out.println("dentro del procA");
           throw new RuntimeException("demo");
       } finally {
           System.out.println("Finally del procA");
       }
    }

   public static void procB() {
       try  {
           System.out.println("dentro del procB");
           return; 7
       } finally {
           System.out.println("finally del procB");
       }
    }

    public static void main(String args[])  {
        try  {
            procA();
        } catch(Exception e) {
            procB();
        }
     }
}
```

12. Indica cuál será la salida del siguiente programa y por qué

```java
public class Siete {
   public static void metodo() {
       try  {
           throw new NullPointerException("demo");
       } catch (NullPointerException e) {
           System.out.println("capturada en método");
           throw e;
       }
    }

    public static void main (String args[])  {
        try   {
            metodo();
        }  catch(NullPointerException e)  {
           System.out.println("capturada en main " + e);
        }
    }
}
```

13. Escribe un programa que juegue con el usuario a adivinar un número. El ordenador debe generar un número entre 1 y 500, y el usuario tiene que intentar adivinarlo. Para ello, cada vez que el usuario introduce un valor, el ordenador debe decirle al usuario si el número que tiene que adivinar es mayor o menor que el que ha introducido el usuario. Cuando consiga adivinarlo, debe indicárselo e imprimir en pantalla el número

de veces que el usuario ha intentado adivinar el número. Si el usuario introduce algo que no es un número, debe indicarlo en pantalla, y contarlo como un intento. Se debe controlar que la lectura que se realiza es realmente un número entero, y en caso contrario, dar un mensaje de error y volver a pedirlo. *Scanner* indica que no ha conseguido reconocer la entrada lanzando la excepción `InputMismatchException`.

14. Intenta adivinar la salida por pantalla que produciría el siguiente programa:

```java
public class EjemploExcepciones {

    public static int devuelveNumero(int num) {
        try {
            if (num % 2 == 0) {
                throw new Exception("Lanzando excepcion");
            }
            return 1;
        } catch (Exception ex) {
            return 2;
        } finally {
            return 3;
        }
    }

    public static void main(String[] args) {
        System.out.println(devuelveNumero(1));
    }
}
```

3. Log4j2

1. Escribe un programa simple en el que usando Log4j2 se impriman por consola TODOS los niveles de error.
2. Escribe un programa simple en el que usando Log4j2 se impriman en SYSTEM_ERR los errores FATAL y el resto de los niveles de error por SYSTEM_OUT.
3. Escribe un programa de ejemplo en el que los errores FATAL vayan a un fichero de texto, los de nivel ERROR aparezcan en rojo por la consola (system_err) y el resto aparezcan por la consola normal (system_out).
4. Cambia el Pattern de salida de manera que lo primero que aparezca en la linea de LOG sean tus iniciales.

4. Fuentes de información

- Wikipedia
- Programación de servicios y procesos - FERNANDO PANIAGUA MARTÍN [Paraninfo]
- Programación de Servicios y Procesos - ALBERTO SÁNCHEZ CAMPOS [Ra-ma]
- Programación de Servicios y Procesos - Mª JESÚS RAMOS MARTÍN - [Garceta] (1ª y 2ª Edición)
- Programación de servicios y procesos - CARLOS ALBERTO CORTIJO BON [Síntesis]
- Programació de serveis i processos - JOAR ARNEDO MORENO, JOSEP CAÑELLAS BORNAS i JOSÉ ANTONIO LEO MEGÍAS [IOC]
- GitHub repositories:
    - https://github.com/ajcpro/psp
    - https://oscarmaestre.github.io/servicios/index.html
    - https://github.com/juanro49/DAM/tree/master/DAM2/PSP
    - https://github.com/pablohs1986/dam_psp2021
    - https://github.com/Perju/DAM
    - https://github.com/eldiegoch/DAM
    - https://github.com/eldiegoch/2dam-psp-public
    - https://github.com/franlu/DAM-PSP
    - https://github.com/ProgProcesosYServicios
    - https://github.com/joseluisgs
    - https://github.com/oscarnovillo/dam2_2122
    - https://github.com/PacoPortillo/DAM_PSP_Tarea02_La-Cena-de-los-Filosofos