Practica 9

Ejercicio 1: ¿Explique claramente a qué se denomina excepción?

Ejercicio 1-

Una excepción es una condición inesperada o inusual que se da durante la ejecución de un programa y no puede ser manejada en el contexto local.

Ejercicio 2: ¿Qué debería proveer un lenguaje para el manejo de las excepciones? ¿Todos los lenguajes lo proveen?

Ejercicio 2-

Los lenguajes deben proveer como mínimo de:

- Un modo de definirlas.
- Una forma de reconocerlas.
- Una forma de lanzarlas y capturarlas.
- Una forma de manejarlas especificando el código y respuestas.
- Un criterio de continuación.

No todos los lenguajes proveen estos requisitos para el controlamiento de excepciones.

Ejercicio 3: ¿Qué ocurre cuando un lenguaje no provee manejo de excepciones? ¿Se podría simular?
Explique cómo lo haría

Ejercicio 3-

Que un lenguaje no provea de excepciones no significa que no se pueda manejar errores. El programador debe implementar estrategias manuales para controlar el flujo cuando algo falla.

Ejercicio 4: Cuando se termina de manejar la excepción, la acción que se toma luego es importante.
Indique

1. ¿Qué modelos diferentes existen en este aspecto?

2. Dé ejemplos de lenguajes que utilizan cada uno de los modelos presentados anteriormente. Por
   cada uno responda respecto de la forma en que trabaja las excepciones.
   a. ¿Cómo se define?
   b. ¿Cómo se lanza?
   c. ¿Cómo se maneja?
   d. ¿Cuál es su criterio de continuación?
3. ¿Cuál de esos modelos es más inseguro y por qué?

Ejercicio 4-

1. Para la acción a realizar post terminar de manejar una excepción hay dos modelos. El modelo de **reasunción** donde se detecta la excepción, se maneja y se continua con la ejecución desde la siguiente linea donde ocurrió. Por otro lado tenemos el modelo de **terminación** donde se busca un manejador y luego no se sigue la ejecución normal sino que termina la ejecución de la unidad donde se produjo.
2. Ejemplo de Reasunción → PL1:
   - Se define con la palabra clave CONDITION.
   - Se lanza con la palabra clave SIGNAL.
   - Se maneja con la sentencia ON CONDITION (nombre de la excepción).
   - Su criterio de continuación es que el manejador devuelve el control a la sentencia siguiente de donde se levantó.

   Ejemplo de Terminación → JAVA
   - Se define con una instancia de la clase Throwable.
   - Se lanza con a través de throw.
   - Se maneja mediante 5 palabras: try, catch, throw, throws, finally.
   - Su criterio de continuación es por terminación.
3. Es más inseguro el modelo de terminación ya que evita repetir errores críticos, facilita el lanzamiento y requiere de menor complejidad y riesgo de inconsistencias.

Ejercicio 5: La propagación de los errores, cuando no se encuentra ningún manejador asociado, no se

implementa igual en todos los lenguajes. Realice la comparación entre el modelo de Java, Python y PL/1, respecto a este tema. Defina la forma en que se implementa en un lenguaje conocido por Ud.

Ejercicio 5

> PL1 → Si no se encuentra un manejador en el ámbito donde se encuentra se propaga estáticamente, es decir, el proceso se repite para las sentencias incluidas estáticamente. Y si se encuentra en el ámbito se ejecuta ahí.
>
> JAVA → La propagación es automática y jerárquica. Si hay un catch en el método actual, la excepción se propaga al método que lo llamó, y así sucesivamente. Si no se encuentra un manejador en toda la pila el programa se termina abruptamente y se imprime el stack trace detallado.
>
> PYTHON → La propagación es primero estática, luego dinámica y por último se produce un error.

Ejercicio 6: Sea el siguiente programa escrito en Pascal
¿Qué modelo de manejo de excepciones está simulando? ¿Qué necesitaría el programa para que
encuadre con los lenguajes que no utilizan este modelo? Justifique la respuesta.

```
Procedure Manejador;
Begin ... end;
Procedure P(X:Proc);
begin
....
if Error then X;
....
end;
Procedure A;
begin
....
P(Manejador);
....
end;
```

Ejercicio 6

El manejo de excepciones que está simulando es del modelo de reasunción ya que luego de terminar la ejecución del manejador sigue ejecutando líneas. Lo que necesitaría el programa para que encuadre con los lenguajes que no utilizan este modelo es una instrucción para interrumpa la ejecución normal, por ejemplo **exit**, **throw**, **raise**, etc.

Ejercicio 7: Sea el siguiente programa escrito en Pascal:

```
Program Principal;
var x:int;  b1,b2:boolean;
 Procedure P (b1:boolean);
      var x:int;
      Procedure Manejador1
                        begin
                  x:=x + 1;
                end;
       begin
       x:=1;
       if b1=true then Manejador1;
       x:=x+4;
       end;
       Procedure Manejador2;
         begin
               x:=x * 100;
       end;
```

```
Begin
      x:=4;
      b2:=true;
      b1:=false;
      if b1=false then Manejador2;
      P(b);
      write (x);
End.
```

a) Implemente este ejercicio en PL/1 utilizando manejo de excepciones
b) ¿Podría implementarlo en JAVA utilizando manejo de excepciones? En caso afirmativo, realícelo.

Ejercicio 7-

```
MAIN: PROCEDURE OPTIONS(MAIN);
  DECLARE x FIXED BINARY;
  DECLARE b1, b2 BIT(1);

  ON ERROR BEGIN;
     CALL Manejador2;
  END;

  /* Procedimiento P con parámetro b1 */
  P: PROCEDURE(p_b1 BIT(1));
     DECLARE p_b1 BIT(1);
     DECLARE x FIXED BINARY;
```

```
      ON ERROR BEGIN;
         CALL Manejador1;
      END;

      x = 1;
      IF p_b1 = '1'B THEN CALL Manejador1;
      x = x + 4;
   END P;

   Manejador1: PROCEDURE;
      x = x + 1;
   END Manejador1;

   Manejador2: PROCEDURE;
      x = x * 100;
   END Manejador2;

   /* Código principal */
   x = 4;
   b2 = '1'B;      /* true */
   b1 = '0'B;      /* false */

   IF b1 = '0'B THEN CALL Manejador2;
   CALL P(b2);
   PUT SKIP LIST('x = ', x);

END MAIN;
```

Ejercicio 8: Sean los siguientes, procedimientos de un programa escrito en JAVA:

```
public static void main (String[] argos){
        Double array_doubles[]= new double[500];
        for (int i=0; i<500; i++){
                array_doubles[i]=7*i;
         }
        for (int i=0 ; i<600 ; i=i+25){
              try{
                    system.out.println("El elemento en "+ i + " es " + acceso_por_indice (array_doubles,i));
              }
              catch(ArrayIndexOutOfBoundsException e){
                    system.out.println(e.tostring());
              }
              catch(Exception a){
                    system.out.println(a.tostring());
              }
              finally{
                    system.out.println("sentencia finally");
              }
        }
}

Public static double acceso_por_indice (double [] v, int indice) throws Exception; ArrayIndexOutOfBoundsException{
          if ((indice>=0) && (indice<v.length)){
              Return v[indice];
          }
          else{
                if (indice<0){
                  // caso excepcional
                  Throw new ArrayIndexOutOfBoundsException(" el índice" + indice + " es un número negativo");
```

```
                }
                else{
                  // caso excepcional
                  Throw new Exception(" el indice" + indice + " no es una posición válida");
                  }
          }
}
```

a) Analizar el ejemplo y decir qué manejadores ejecuta y en qué valores quedan las variables. JUSTIFIQUE LA RESPUESTA.
b) La excepción se propaga o se maneja en el mismo método? ¿Qué instrucción se agrega para poder propagarla y que lleve información?.
c) como modificaría el método "acceso_por_indice" para que maneje él mismo la excepción.

Ejercicio 8-

a. Los manejadores que ejecuta son:

   - Try, catch y finally en el segundo for del programa.

   Variables:

   - El bucle usa `i = 0` hasta `i = 600` en pasos de 25 → los valores de `i` serán: `0, 25, 50, 75, ..., 600`.
   - Cuando `i >= 500`, se lanza `ArrayIndexOutOfBoundsException`, capturada en el primer `catch`.

b.

La excepción se propaga desde el método `acceso_por_indice` hacia el método `main` porque este método declara que puede lanzar excepciones usando `throws Exception`. Para propagar una excepción en Java, se usa la palabra clave `throws` en la declaración del método. Esto permite que el método no maneje directamente la excepción, sino que la pase al método que lo invocó.

c.

```
public static double acceso_por_indice(double[] v, int indice) {
    try {
        if (indice >= 0 && indice < v.length) {
            return v[indice];
        } else if (indice < 0) {
            throw new ArrayIndexOutOfBoundsException("El índice " + indice +
        } else {
            throw new Exception("El índice " + indice + " no es una posición vá
        }
    } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println("Excepción: " + e.toString());
    } catch (Exception e) {
        System.out.println("Excepción: " + e.toString());
    } finally {
        System.out.println("sentencia finally");
    }

    // Devuelve un valor por defecto si hubo excepción
    return Double.NaN;  // NaN: Not a Number
}
```

Ejercicio 10: ¿Qué modelo de excepciones implementa Ruby?. ¿Qué instrucciones específicas provee ellenguaje para manejo de excepciones y cómo se comportan cada una de ellas?

Ejercicio 10: Ruby implementa un **modelo de excepciones basado en clases**, similar al de otros lenguajes orientados a objetos como Java o Python. En Ruby, todas las excepciones son objetos que heredan de la clase base `Exception`. El manejo de excepciones se realiza mediante bloques `begin-rescue-end`, lo cual permite capturar errores y manejar situaciones excepcionales sin interrumpir el flujo del programa.

Instrucciones específicas para el manejo de excepciones en Ruby y su comportamiento:

1. **`begin ... rescue ... end`**

   Se utiliza para capturar y manejar excepciones.

   ```
   ruby
   CopiarEditar
   begin
     # código que puede lanzar una excepción
   rescue NombreDeExcepcion ⇒ e
     # código para manejar esa excepción
   end
   ```

   Si ocurre una excepción dentro del bloque `begin`, Ruby busca un bloque `rescue` que pueda manejarla.

2. **`rescue`**

   Se pueden usar múltiples cláusulas `rescue` para manejar distintos tipos de excepciones:

   ```
   ruby
   CopiarEditar
   begin
     # ...
   rescue ZeroDivisionError
     puts "No se puede dividir por cero."
   rescue StandardError ⇒ e
     puts "Ocurrió un error: #{e.message}"
   ```

```
end
```

3. `else`

   Se ejecuta **solo si no ocurre ninguna excepción** en el bloque `begin`.

```
ruby
CopiarEditar
begin
  puts "Intento exitoso"
rescue
  puts "Hubo un error"
else
  puts "No hubo errores"
end
```

4. `ensure`

   Es equivalente al `finally` en otros lenguajes. Se ejecuta **siempre**, ocurra o no una excepción. Se usa generalmente para liberar recursos.

5. `raise`

   Permite lanzar una excepción manualmente.

```
ruby
CopiarEditar
raise "Ocurrió un error"
raise ArgumentError, "Argumento inválido"
```

Ejercicio 11: Indique el mecanismo de excepciones de javascript.

Ejercicio 11: JavaScript utiliza un mecanismo de manejo de excepciones basado en bloques `try`, `catch`, `finally` y la instrucción `throw`. Este modelo es similar al de otros lenguajes como Java o Ruby, y permite capturar errores durante la ejecución del programa para evitar que este termine abruptamente.

Ejercicio 4