OCA Java 8 Exam - ERRORES COMUNES

JAVA BASICO

- **Prestar atencion a las trampas en código!!!!**
- ERROR DE COMPILACIÓN EN LINEA 3: El parámetro del metodo main tiene el mismo nobre que el array de int

```
1:public class Whizlabs {
2:   public static void main(String [ ] a) {
3:       int [ ] a = {1,2,3};
4:       System.out.print(a[3]);
5:   }
6:}
```

- **Prestar atencion a las trampas en código!!!!**
- ERROR DE COMPILACIÓN EN LINEA 6: La variable **x** es local al bloque **for**

```
1:public class Whizlabs {
2:
3:  public static void main(String[] args) {
4:
5:    for(int x = 3;x>-1;x--);
6:      System.out.print(x);
7:  }
8:}
```

- El siguiente es un literal de tipo **long** válido:

```
long x = 0x99ffCl;
```

- Solo las clases ***public*** y ***default*** pueden ser **top level classes**
    - A: class one(){}
    - B: static class Test{}
    - C: protectec class Test(){}
    - **D: class If{}**
- Cual de los siguientes es obligatorio en una clase?
    - A: A method
    - B: A variable
    - **C: A constructor**
    - D: a main method

- Este código genera que se lance un **Error** en tiempo de ejecución, ya que no existe un metodo **main**:

```
1:public class Whizlabs {
2:     static
3:        {
4:          System.out.println("This java program will run without the main
5:          System.exit(0);
6:        }
7:}
```

- Este programa da error de compilación porque no existe un metodo **print()** sin parámetros

```
public class Whizlabs {
   public static void main(String[] args) {
       System.out.print();
   }
}
```

ARRAYS

- Este código genera **NegativeArraySizeException**

```
int array[] = new int [-2];
```

- Error de compilación en linea 4: Solo se puede usar un bloque de inicialización de arrays, por ejemplo: **{"1", "2"}** cuando se define el array

```
1:public class Whizlabs {
2:   public static void main(String[] args) {
3:        String[] array;
4:        array = {"1", "2"};
5:        for (String s : array) {
6:            System.out.println("s = " + s);
7:        }
8:   }
9:}
```

- Error de compilación en linea 5: Los indices de los arrays solo pueden ser **int** o **Integer**

```
1:public class Whizlabs {
2:    static long index = 2;
3:    public static void main(String[] args) {
4:        int[][] array = {{}, {1, 2, 3}, {4, 5}};
5:        System.out.println(array[index][1]);
6:    }
7:}
```

- ERROR EN LINEA 4: Este programa lanza **NullPointerException** en tiempo de ejecución, ya que tratamos de inicializar los elementos del array, sin crear el array

```
1:public class Whizlabs {
2:   public static void main(String[] args) {
3:       int array[][] = new int[2][];
4:       array[0][0] = 1;
5:       array[0][1] = 2;
6:       array[0][2] = 3
7:   }
8:}
```

Forma correcta:

```
1:public class Whizlabs {
2:   public static void main(String[] args) {
3:       int array[][] = new int[2][];
4:       array[0] = new int[3];
5:       array[0][0] = 1;
6:       array[0][1] = 2;
7:       array[0][2] = 3
8:   }
9:}
```

- Este programa imprime 9

```
1:public class Whizlabs {
2:   public static void main(String[] args) {
3:       int array[] = {2, 5, 9, 5, 0, 3};
4:       Arrays.sort(array, 2, 6);
5:       System.out.println(array[2] + array[5]);
6:   }
7:}
```

EXPLICACION:

- **Arrays.sort(int[] a, int fromIndex, int toIndex)** incluye el parametro **fromIndex** y EXCLUYE el parametro **toIndex**, entonces, en la linea 4 sólo ordena los indices entre el 2 y el 5.
- El array quedaría así: {2, 5, 0, 3, 5, 9}
- array[2] + array[5] devuelve 9, porque array[2] = 0 y array [5] = 9
- Este programa imprime **10**, ya que en la linea 5 primero uso el valor de **i** y después lo incremento.

```
1:public class Whizlabs {
2:   static int i;
3:   public static void main(String[] args) {
4:       int array[] = {10, 11, 12};
5:       System.out.println(array[i++]);
6:   }
7:}
```

- Este programa imprime: **\t&1LNOSaÇç**, porque los caracteres, se ordenan de acuerdo a la tabla **ASCII**

```
import java.util.Arrays;

public class Whizlabs {
    public static void main(String[] args) {
        String[] a = {"N", "L", "a", "O", "S", "1", "\t", "&", "ç", "Ç"};
        Arrays.sort(a);
        for (String as : a) {
            System.out.print(as);
        }
    }
}
```

```
    &1LNOSaÇç
```

- El numero total de enteros que este array puede almacenar es **8**

```
1: int nums[][] = new int[3][3];
2: nums[0] = new int[2];
```

EXPLICACION:

1. En la línea 1 se declara un array de **9** elementos:

0	0	0
0	0	0
0	0	0

2. En la línea 2 se declara un array de **2** elementos en la posición **0** de **nums**:

0	0	
0	0	0
0	0	0

- Este programa imprime **010**

```
public class Whizlabs {
    public static void main(String[] args) {
        final int[] ints = new int[3];
        int len = ints.length;
        ints[1]++;
        for (int i : ints) {
            System.out.print(i);
        }
    }
}
```

- Error de compilación en linea 5: No se puede asignar un array constante a un array ya definido. Sólo es válido cuando se define el array.

```
1:public class Whizlabs {
2:    public static void main(String[] args) {
3:        int[][] ints = new int[3][2];
4:        ints[0] = new int[3];
5:        ints[2] = {1, 2, 3};
6:        System.out.print(ints[0].length + ints[2].length);
7:    }
8:}
```

- Este programa arroja un **NullPointerException** en el **Arrays.sort()**, ya que no se inicializó ints[1]

```
public class Whizlabs {
    public static void main(String[] args) {
        int[][] ints = new int[2][];
        Arrays.sort(ints[1]);
        System.out.print(Arrays.toString(ints[1]));
    }
}
```

ENCAPSULACION

- Este programa imprime **5**

```
public class Whizlabs {
    static int x = 1;

    public Whizlabs() {
        x++;
    }

    public static void main(String[] args) {
        System.out.println(x+ check(2));
    }

    static int check(int i) {
        return new Whizlabs().x * i;
    }
}
```

Explicación:

EXCEPTIONS

- Error de compilación en linea 3: No lanzamos ni manejamos una **checked exception** con **try-catch**

```
1:public class Whizlabs {
2:    public static void main(String[] args) {
3:        new Whizlabs().method();
4:    }
5:
6:    public void method() throws Exception {
7:        System.out.println("Hola");
8:    }
9:}
```

- Error de compilación en linea 5: IOException de un bloque **try-catch** no se lanza nunca

```
1:public class Whizlabs {
2:    public static void main(String[] args) {
3:        try {
4:            method();
5:        } catch (IOException e) {
6:        }
7:    }
8:
9:    public static void method() {
```

```
10:         System.out.println("Hola");
11:     }
12:}
```

- Error de compilación en lineas 7, 9, 11: Cuando usamos un bloque multicatch, la variable de excepción es implicitamente **final**. Entonces, al intentar asignarle un valor diferenta a la variable de excepción, se produce un error de compilación.

```
 1:public class Whizlabs {
 2:     public static void main(String[] args) {
 3:         try {
 4:             System.out.println(args[0]);
 5:         } catch (ArrayIndexOutOfBoundsException | ArithmeticException | Nu
 6:             if (e instanceof ArrayIndexOutOfBoundsException) {
 7:                 e = new ArrayIndexOutOfBoundsException("Out of bounds");
 8:             } else if (e instanceof NullPointerException) {
 9:                 e = new NullPointerException("Null value");
10:             } else {
11:                 e = new ArithmeticException("Aritmetic");
12:             }
13:             System.out.println(e.getMessage());
14:         }
15:     }
16:}
```

- Error de compilación en linea 19: El bloque **try** externo no tiene bloques **catch** o **finally**

```
 1:public class Whizlabs {
 2:     public static void main(String[] args) {
 3:         try {
 4:             int x = args.length;
 5:             int v = 10 / x;
 6:             System.out.print(x);
 7:             try {
 8:                 if (x == 1)
 9:                     x = x / x - x;
10:                 if (x == 2) {
11:                     int[] c = {2};
12:                     c[3] = 3;
13:                 }
14:             } catch (ArrayIndexOutOfBoundsException e) {
15:                 System.out.println("out of bounds.");
16:             } catch (ArithmeticException e) {
17:                 System.out.println("Arithmetic");
18:             }
19:         }
20:     }
21:}
```

LOOPS

- El siguiente programa imprime **0** sin terminar nunca

```
1:public class Whizlabs {
2:   public static void main(String[] args) {
3:       int x = 0;
4:       while ((x = 0) <= 1) {
5:           System.out.print(x);
6:           x++;
7:       }
8:   }
9:}
```

- El siguiente programa imprime **Mayor que 10** porque:
  - Las variable **int** tienen valores entre -2147483648 y 2147483647.
  - Si el valor de ina variable de tipo **int** alcanza su valor mínimo, y se intenta restar, la misma pasará a su valor **máximo** (2147483647).

```
public class Whizlabs {
    public static void main(String[] args) {
        int y = 0;
        while (y-- < 10) {
            continue;
        }
        String message = y > 10 ? "Mayor que " : "Menor que";
        System.out.println(message + "10");
    }
}
```

- Error de compilación en linea 7: Debido a la instruccion **continue** el código de la linea 7 es inalcanzable, por lo que el compilador arroja una excepción.

```
 1:public class Whizlabs {
 2:    public static void main(String[] args) {
 3:        String[] str = {"A", "B", "C"};
 4:
 5:        for (String s : str) {
 6:            continue;
 7:            System.out.println(s);
 8:        }
 9:    }
10:}
```

- Este programa

```
public class Whizlabs {
    public static void main(String[] args) {
        for (int x = 0; x < 10; x++) {
            if (5 == x) {
                break;
            }
            System.out.print(x + " ");
        }
        System.out.println();
        for (int x = 0; x < 10; x++) {
            if (5 == x) {
                continue;
            }
            System.out.print(x + " ");
        }
    }
}
```

- Imprime:

```
0 1 2 3 4
0 1 2 3 4 6 7 8 9
```

OPERADORES Y DECISIONES

- Este programa devuelve **20**

```
 1:public class Whizlabs {
 2:    public static void main(String[] args) {
 3:        int y = 10;
 4:
 5:        if (y++ == 10)
 6:            if (y-- == 10)
 7:                if (y == 10);
 8:                else y *= 3;
 9:            else y *= 2;
10:
11:        System.out.println(y);
12:    }
13:}
```

- Error de compilación en linea 10: La variable **str** no ha sido inicializada

```
 1:public class Whizlabs {
 2:    static boolean b = false;
 3:
 4:    public static void main(String[] args) {
 5:        String str;
 6:
 7:        if (b = true) {
 8:            str = "true";
 9:        }
10:        str = b ? str : "false";
11:        System.out.println(str);
12:    }
13:}
```

- Este programa lanza **NullPointerException** en la linea 7: La variable y no ha sido inicializada.

```
 1:public class Whizlabs {
 2:    static Integer y;
 3:
 4:    public static void main(String[] args) {
 5:        int x = 10;
 6:
 7:        if (x++ > 10 & y++ == 1) y += 10;
 8:        System.out.print(y);
 9:    }
10:}
```

- Este programa:

```
public class Whizlabs {
    public static void main(String[] args) {
        int x = -10;
        int y = 10;

        System.out.println("if(y++ > 10 | x % (-3) == 1)  : " + (y++ > 10 | x
        System.out.println("if(y >= 10 & x % (-3) == -1)  : " + (y >= 10 & x
        System.out.println("if(y > 10 | x % (3) == -1)    : " + (y > 10 | x %
        System.out.println("if(++y > 10 & x % (-3) == 1)  : " + (++y > 10 & x
        System.out.println("if(++y > 10 | x % (-3) == 1)  : " + (++y > 10 | x
        System.out.println("if(y++ >= 10 ^ x % (-3) == -1): " + (y++ >= 10 ^
    }
}
```

Imprime:

```
if(y++ > 10 | x % (-3) == 1)  : false
if(y >= 10 & x % (-3) == -1)  : true
if(y > 10 | x % (3) == -1)    : true
if(++y > 10 & x % (-3) == 1)  : false
if(++y > 10 | x % (-3) == 1)  : true
if(y++ >= 10 ^ x % (-3) == -1): false
```

TIPOS DE DATOS

- Este programa imprime **36** porque:
  - SIZE: Es el numero de bits necesarios para representar un int
  - BYTES: Es el numero de bytes necesarios para representar un int

```
1:public class Whizlabs {
2:   public static void main(String[] args) {
3:       Integer x = 8;
4:       System.out.println(x.SIZE + x.BYTES);
5:   }
6:}
```

GARBAGE COLLECTOR

- El objeto creado en la linea **5** se libera es elegible por el **garbage collector** en la linea **10**

```
 1:public class Employee {
 2:   private static Employee e;
 3:
 4:   public static void main(String[] args) {
 5:       Employee e1 = new Employee();
 6:       Employee e2 = new Employee();
 7:       e2.join(e1);
 8:       e1 = new Employee();
 9:       e1 = null;
10:       e2.join(e1);
11:    }
12:
13:    public void join(Employee s) {
14:        e = s;
15:    }
16:}
```

- Explicación:
  - Un objeto es elegible para ser eliminado por el garbage collector en alguno de estos dos casos:
    1. El objeto no tiene ninguna referencia apuntando a él
    2. Todas las referencias al objeto quedaron fuera de ámbito
  1. En la linea 5 se crea un objeto de tipo Employee en la variable **e1**
  2. En la linea 6 se crea otro objeto Employee en la variable e2
  3. En la linea 7 se llama al metodo **join**, que pasa la variable e1 por referencia.
  4. El metodo join ejecuta la instrucción **e = s;**.
     - Existen **dos referencias** al objeto creado en linea 5, variables **e** y **s**
  5. En linea 8 se crea otro objeto Employee y se referencia con la variable **e1**
     - Existe sólo **una referencia** al objeto creado en la linea 5, variable **e**
  6. En la linea 10 se llama al metodo **join** pasando la variable **e1** que es **null**
     - La variable de referencia **e** es nula.
     - En este punto, ya no existe referencia al objeto creado en la linea 5. Entonces, dicho objeto es elegible por el garbage collector

API de Java

ArrayList

toArray

- Copia los elementos de un arraylist en un array.
  1. Si el array destino pasado como parámetro es más chico que el arraylist, devuelve un nuevo array con todos los elementos del arraylist.

```
public class Whizlabs {
    public static void main(String[] args) {
        List<String> strings = new ArrayList<>();
        strings.add("A");
        strings.add("B");
        strings.add("C");

        String[] arrayChico = new String[2];
        arrayChico = strings.toArray(arrayChico);

        for (String s : arrayChico) {
            System.out.print(s + " ");
        }
    }
}
```

Devuelve:

```
A B C
```

2. Si el array destino pasado como parámetro es más grande que el arraylist, devuelve el destino con todos los elementos del arraylist, y los lugares restantes del array destino en **null**

```
public class Whizlabs {
    public static void main(String[] args) {
        List<String> strings = new ArrayList<>();
        strings.add("A");
        strings.add("B");
        strings.add("C");

        String[] arrayGrande = new String[5];
        arrayGrande = strings.toArray(arrayGrande);

        for (String s : arrayGrande) {
            System.out.print(s + " ");
        }
    }
}
```

Devuelve:

```
A B C null null
```

ArrayList sin especificar tipo, y List sin especificar tipo

- Si se crea una **List** sin especificar su tipo, con un **new ArrayList<>()** se genera una lista de **Object**

```
public class Whizlabs {
    public static void main(String[] args) {
        List list = new ArrayList<>();
        list.add(1);
        list.add(4);
        list.add("A");
        for (Object o :
                list) {
            System.out.println(o + " -> " + o.getClass());
        }
    }
}
```

Devuelve:

```
1 -> class java.lang.Integer
4 -> class java.lang.Integer
A -> class java.lang.String
```

- ERROR DE COMPILACIÓN EN LINEA 6: Se invoca a **Integer.max** pasandole un **Object**

```
1:public class Whizlabs {
2:   public static void main(String[] args) {
3:       List list = new ArrayList<>();
4:       list.add(1);
5:       list.add(4);
6:       System.out.println(Integer.max(list.get(0),1));
7:   }
8:}
```

String

- Este programa imprime **true false**

```
public class Whizlabs {
    public static void main(String[] args) {
        String s1 = "Cadena";
        String s2 = new String("Cadena");

        System.out.print(s1.equals(s2) + " ");
        System.out.print(s1 == s2);
    }
}
```

StringBuilder

- El siguiente programa imprime **The latestJavaversion is 1.8**

```
1:public class Whizlabs {
2:    public static void main(String[] args) {
3:        StringBuilder s = new StringBuilder("Java");
4:        s.insert(0, "The latest").append("version is")
5:         .append(" 1.7").delete(27, 28)
6:         .append("8").substring(4);
7:        System.out.print(s);
8:    }
9:}
```

- Explicación:
    1. En la linea 3 se crea un objeto StringBuilder con contenido "Java".
    2. En la línea 4, los métodos se ejecutan de izquierda a derecha.
    3. El método **insert** agrega "The latest" en el índice cero.
        - Ahora el contenido es "The latestJava".
    4. **append** agrega "version is" al final.
        - Ahora el contenido es "The latestJavaversion is".
    5. El siguiente ***append** agrega "1.7" al final.
        - Ahora el contenido es "The latestJavaversion is 1.7".
    6. El método de **delete** elimina el carácter "7".
        - Ahora el contenido es "The latestJavaversion is 1.".
    7. El siguiente **append** agrega "8" al final.
        - Ahora el contenido es "The latestJavaversion is 1.8".
    8. El método **substring** devuelve una Cadena a partir del índice "4" pero no cambia el contenido del objeto StringBuilder en la Línea 3. Ese valor no se asigna a ninguna variable de referencia, por lo que será recolectado como basura.
    9. Así que el resultado final es **The latestJavaversion is 1.8**

- El siguiente programa lanza **ArrayIndexOutOfBoundsException**

```
public class Whizlabs {
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder("Whizlab");
        char[] ch = new char[4];
        sb.getChars(1, 5, ch, 1);
        for (char c : ch)
            System.out.println(c);
    }
}
```

- Explicación:

    1. El metodo getChars es así:
    2. **public void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)**
    3. El primer caracter a copiar es al srcBegin
    4. El ultimo caracter a copiar es el srcEnd -1
    5. Los caracteres se copian en el destino empezando en dstBegin
    6. Los caracteres se terminan de copiar en el destino en el indice: dstBegin + (srcEnd - srcBegin) -1
    7. En el ejemplo, esto es: 1 + (5 - 1) - 1 = 4
    8. Como el array ch es de 4 elementos, su mayor indice es **3**. Por eso lanza **ArrayINdexOutOfBoundsException**

- El siguiente programa imprime **1Zo-808**

```
public class Whizlabs {
    public static void main(String[] args) {
        char[] chars = {'1', 'Z', 'o', '-', '8', '1'};
        StringBuilder sb = new StringBuilder();
        sb.append(chars, 0, chars.length - 1);
        sb.append("0");
        sb.append("8");
        System.out.print(sb);
    }
}
```

- Explicación:
    1. La version sobrecargada del método **append** de **StringBuilder** es:
    2. public StringBuilder append(char[] str, int offset, int len)
    3. Donde **offset** y **len** son lis *indices* de inicio y fin del array a ser agregados al StringBuilder
    4. sb.append(chars, 0, chars.length - 1) --> agregará **"1Zo-81"**

- El siguiente programa imprime **Whizlab**

```
1:public class Whizlabs {
2:   public static void main(String[] args) {
3:       StringBuilder sb = new StringBuilder("Whiz");
4:       sb = sb.append("lab");
5:       sb.append("s");
6:       sb.setLength(7);
7:       System.out.println(sb);
8:   }
9:}
```

- Explicación:
    1. StringBuilder no es inmutable, entonces la asignacion de linea 4 aunque correcta, es innecesaria
    2. Después de ejecutarse la linea 5, sb contiene **Whizlabs**
    3. el método **setLength(7)** corta el contenido de sb después del 7º caracter

LocalDate, LocalTime, LocalDateTime, Period

- PERIOD: Este programa

```
import java.time.Period;

public class Whizlabs {
    public static void main(String[] args) {
        Period p1 = Period.ofYears(1);
        Period p2 = Period.of(0, 1, 0);
        Period p3 = p1.plus(p2);
        System.out.println("years:  " + p3.getYears());
        System.out.println("months: " + p3.getMonths());
        System.out.println("days:   " + p3.getDays());
    }
}
```

Devuelve:

```
years:  1
months: 1
days:   0
```

porque:

- **getYears:** devuelve la cantidad de años del período.
- **getMonths:** devuelve la cantidad de meses del período.

- **getDays:** devuelve la cantidad de dias del período. En este caso tenemos un periodo de 1 año, 1 mes, y 0 días

LAMBDAS

- La linea 7 es incorrecta, ya que al faltarle el **;** al final de la linea, no compila. Puede solucionarse así:
  - Agregar el **;** al final de la línea.
  - Eliminar las **{}**, dado que:
    - El cuerpo de un lambda puede ser una **expresion simple**, o tener un **bloque de instrucciones**. Un **bloque de instrucciones** DEBE tener una instruccion **return**, pero dado que **return** NO ES UNA EXPRESION, dicho bloque además debe estar entre **{}**.

```
 1:public class Whizlabs {
 2:    interface Runnable {
 3:        public void run();
 4:    }
 5:
 6:    public static void main(String[] args) {
 7:        Runnable run1 = () -> {System.out.println("Run");}
 8:        Runnable run2 = () -> System.out.println("Run");
 9:    }
10:}
```

Ambito de variables

- Error de compilación en linea 7: Las variables del bucle **for** son válidas solo en el bucle y su cuerpo. La variable **x** no existe en la linea 5, sólo en las lineas **5** y **6**

```
 1:public class Whizlabs {
 2:   public static void main(String[] args) {
 3:       int sum = 0;
 4:
 5:       for(int x = 0;x&lt;=10;x++)
 6:           sum += x;
 7:       System.out.print("Sum for 0 to " + x);
 8:       System.out.println(" = " + sum);
 9:   }
10:}
```

- Error de compilación en lines 9: La variable **number** es válida solo en en bloque **try - catch**.

```
 1:public class Whizlabs {
 2:
 3:    public static void main(String[] args){
 4:        try{
 5:            Double number = Double.valueOf("120D");
 6:        }catch(NumberFormatException ex){
 7:            System.out.println(ex);
 8:        }
 9:        System.out.println(number);
10:    }
11:}
```

- Error de compilación en linea 7: No se pueden usat tipos primitivos para Collections. Se debe usar **Integer** en lugar de **int**

```
 1:import java.util.ArrayList;
 2:import java.util.List;
 3:
 4:public class Whizlabs {
 5:
 6:     public static void main(String[] args){
 7:         List<int> list = new ArrayList<>();
 8:         list.add(21); list.add(13);
 9:         list.add(30); list.add(11);
10:         list.removeIf(e -> e% 2 != 0);
11:         System.out.println(list);
12:     }
13:}
```

- Este programa imprime **I= 3,J= 0**

```
public class Whizlabs {
    static int i;
    int j;

    Whizlabs() {
        j = i++;
    }

    public static void main(String[] args) {
        Whizlabs s = new Whizlabs();
        Whizlabs s1 = new Whizlabs();
        Whizlabs s2 = new Whizlabs();
        System.out.print("I= " + s.i);
        System.out.print(",J= " + s.j);
    }
}
```

Explicación:

- La variable **estática** i, se incrementará con cada llamada instanciación, en los tres **new Whizlabs();**
- La variable j que se imprime en la ultima linea es de la primer instancia, entonces vale **0**
- ERROR DE COMPILACIÓN EN LINEA 2: La variable final **i** no se ha inicializado

```
1:public class Whizlabs {
2:   final int i;
3:
4:   public static void main(String[] args) {
5:       Whizlabs s = new Whizlabs();
6:       System.out.println(s.i);
7:   }
8:}
```

HERENCIA

- Los metodos **static** o **default** de las **interfaces** no pueden sobrescribir metodos de la clase **Object**

```
1:public class Whizlabs {
2:  public static void main(String[] args) {
3:    Interfaz interfaz = new Interfaz(){};
4:  }
5:
6:  interface Interfaz {
7:    default void toString() {
8:        System.out.println("interfaz");
```

```
 9:     }
10:
11:     static boolean equals(Object obj) {
12:         return true;
13:     }
14:  }
15:}
```

- Metodos **default** en las **interfaces**
    1. Los metodos **default** de las **interfaces** deben ser **public**. No pueden ser **private** o **protected**
    2. Los metodos **default** no pueden ser **static**, **final** o **abstract**

Ejemplo:

```
1:public class Whizlabs {
2:  interface Interfaz {
3:    public default void defaultMethod() {
4:        System.out.println("interfaz");
5:    }
6:  }
7:}
```

- Los metodos estaticos de clases padre **no se sobreescriben**

```
public class Whizlabs {
    public static void main(String[] args) {
        Base base = new SubClass();
        base.display();
        base.print();
    }
}

class Base {
    public static void display() {
        System.out.println("static: Base");
    }

    public void print() {
        System.out.println("Print: Base");
    }
}

class SubClass extends Base {
    public static void display() {
        System.out.println("static: SubClass");
    }

    public void print() {
        System.out.println("Print: SubClass");
    }
}
```

Este programa imprime:

```
static: Base
Print: SubClass
```

- No se puede acceder a un metodo privado desde afuera de la clase donde fue definido

```
 1:public class Whizlabs {
 2:   class A {
 3:       private final void print() {
 4:           System.out.println("A");
 5:       }
 6:   }
 7:
 8:   class B extends A {
 9:       private void print() {
10:            System.out.println("B");
11:        }
12:    }
13:
14:    public static void main(String[] args) {
15:        A a = new B();
16:     // Esta linea da error porque se intenta acceder a un metodo privado
17:     // desde afurea de la clase donde está definido
18:        a.print();
19:    }
20:}
```

- Castear un objeto de una superclase a una subclase causa **ClassCastException** en tiempo de ejecución.

```
 1:interface Interfaz {
 2:    void method();
 3:}
 4:
 5:class Clase implements Interfaz {
 6:   public void method() {
 7:       System.out.println("Clase");
 8:   }
 9:}
10:
11:class SubClase extends Clase implements Interfaz {
12:    public void method() {
13:        System.out.println("SubClase");
14:    }
15:}
16:
17:public class Whizlabs {
18:    public static void main(String[] args) {
19:        Clase clase = new Clase();
20:        // Esta linea da ClassCastException en tiempo de ejecucion porque
21:        // se intenta Castear un objeto de la superclase 'Clase' a una
22:        // subclase
23:        SubClase subClase = (SubClase) clase;
```

```
24:         subClase.method();
25:     }
26:}
```

- Error de compilación en linea 13: El constructor de una enum debe ser **private** o bien **default**.

```
 1:public class Whizlabs {
 2:     public static void main(String[] args) {
 3:         System.out.println(Speed.FASTER == Speed.FAST);
 4:     }
 5:
 6:     enum Speed {
 7:         FAST(2),
 8:         FASTER(3),
 9:         SLOW(1);
10:
11:         private final int speed;
12:
13:         public Speed(int code) {
14:             this.speed = code;
15:         }
16:     }
17:}
```

- El método

```
abstract Number number();
```

Puede ser sobreescrito por:

```
public Integer number() throws NumberFormatException;
```