# UD1. IDENTIFICACIÓN de los elementos de un programa INFORMÁTICO

# CUESTIONARIO 2

1. Escribe en orden de **menor a mayor** rango de números enteros los tipos existentes en Java:

|  |  |  |  |
| --- | --- | --- | --- |
| byte | short | int | long |

1. Indica cuál es el tipo primitivo de números reales que permite un mayor rango de números. double
2. Completa la tabla (las celdas no tiene por qué corresponderse en número con los tipos posibles).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Indica los tipos que podrían tener una variable para que fuera asignada a  *float vble* realizando una conversión implícita | | | | | |
| float vble= | long | int | short | byte |  |  |

1. Indica que tendríamos que cambiar para que la declaración de la siguiente variable se convirtiese en la declaración de una constante.

|  |  |
| --- | --- |
| Declaración de una variable | Declaración de una constante |
| int total=34 | Int TOTAL=34; |

1. Señala los errores (si los hay) en las siguientes asignaciones e indica el por qué.

|  |  |
| --- | --- |
| float a = 4.6 | Float a=4.6F (Los float siempre llevan un f al final) |
| byte a = -128 | Correcto |
| double a = 4.6 | Correcto |
| float a = (float) 4.6 | Correcto. Casting de un double a un float. |
| int d = 45L | Int d=45 (La L al final se añade en los tipo long) |
| int d = (int) 45L | Correcto. Casting de un long a un int. |

1. Indica cuál es la afirmación correcta:
   1. El flujo de datos de impresión se representa con la instancia estática **System.out** de la clase **PrintStream**. Incorrecto
   2. El flujo de datos de impresión se representa con el objeto estático **out** de la clase **PrintStream** que es un campo estático de la clase **System**. Correcto
   3. El flujo de datos de impresión se representa con la instancia de **System** y su método **out**. Incorrecto
   4. El flujo de datos de impresión se representa con el paquete **System.out**. Incorrecto
2. Completa la afirmación siguiente:

Los datos que fluyen a través de ordenador desde una entrada hacia una salida o viceversa se denomina flujo de datos o *stream*.

1. Si la clase **System** pertenece al paquete al paquete **java.lang** y la clase **Scanner** pertenece al paquete **java.util**, indica cuál de las dos clases necesita que ser importada de forma explícita. Scanner
2. Completa el método de la clase Scanner que falta en el siguiente código:

int numero**;**

Scanner input **=** **new** Scanner**(**System**.**in**);**

System**.**out**.**println**(**"introduzca un numero"**);**

numero **=** input**.nextInt();**

**if** **(**numero **%** 2 **==** 0**)** **{**

System**.**out**.**println**(**"el numero " **+** numero **+** " es par."**);**

**}** **else** **{**

System**.**out**.**println**(**"el numero " **+** numero **+** " es impar."**);**

**}**

1. De los siguientes errores, indica cuáles son de compilación y de ejecución:

|  |  |  |
| --- | --- | --- |
| Errores | De ejecución | De compilación |
| No poner un ; al final de una sentencia. |  | x |
| Introducir un valor entero fuera de rango en una entrada por teclado. |  | x |
| Utilizar un nombre reservado para nombrar a una variable. |  | x |
| Asignar a una variable un literal de un tipo no compatible. |  | x |
| Realizar una operación de división por 0. | x |  |
| Intentar acceder a los datos de un fichero que no existe. | x |  |

1. Indica con una cruz las afirmaciones que son correctas:

|  |  |
| --- | --- |
| Afirmación | Cierta |
| Una excepción es un evento que se produce ante un error en tiempo de ejecución | X |
| Una excepción es un evento que se produce ante un error en tiempo de compilación |  |
| Cuando se produce una excepción en un programa éste se interrumpe y muestra en la consola información sobre la excepción producida. | X |
| Si queremos que no se interrumpa el programa al producirse una excepción debemos utilizar el bloque *try/catch*. Este bloque evitará que el programa se interrumpa. Las consecuencias serán: se saltará la instrucción que provoque la excepción, continuando la ejecución en la siguiente instrucción a la que produjo la excepción, al finalizar el bloque *try* pasará a ejecutarse el bloque *catch*. |  |
| Si queremos que no se interrumpa el programa al producirse una excepción debemos utilizar el bloque *try/catch*. Este bloque evitará que el programa se interrumpa. Las consecuencias serán: no se ejecutarán las instrucciones siguientes a la que produjo la excepción dentro del bloque *try* y la ejecución saltará al contenido del bloque *catch*. | X |
| Las excepciones se tratan con clases específicas y todas ellas derivan de la clase *ExceptionExecution*. |  |
| Las excepciones se tratan con clases específicas y todas ellas derivan de la clase *Exception*. | X |

1. Indica qué tendríamos que añadir al siguiente código para que se ejecutase ***instrucciónA*** independientemente de que se produzca una excepción el bloque try o no.

**try** **{**

//Instrucciones del bloque try

**}** **catch(Exception** e**)** **{**

//Instrucciones del bloque catch

**}**

InstruccionA

**try** **{**

//Instrucciones del bloque try

**}** **catch(Exception** e**)** **{**

//Instrucciones del bloque catch

**}**

1. Analiza el siguiente código y escríbelo de nuevo para resolver los problemas que puede generar. Explica dónde está el problema y por qué se produce.

**int[] array = new int[20];**

**try {**

**array[20] = 24;**

**int b = 0;**

**int a = 23 / b;**

**} catch (Exception excepcion) {**

**System.out.println(" Error desconocido");**

**} catch (ArrayIndexOutOfBoundsException excepcion) {**

**System.out.println(" Error de índice en un array");**

**} catch (ArithmeticException excepcion) {**

**System.out.println(" Error aritmético");**

**}**

**Va a salir “Error de índice en un array” porque lestá definido de tamaño 20, es decir, tiene valores asignados en los puestos de 0 a 19 (20 en total).**

**Así que el mayor array[n], es con n=19.**

**Si lo cambiamos y le ponemos algo con sentido, el error que nos detecta es el siguiente: el de la operación en la que dividimos entre 0 ya que estamos usando el tipo int.**

**Reescribiendo el código:**

**int[] array = new int[20];**

**array[19] = 24;**

**float b = 0f;**

**float a = 23f / b;**

1. ¿Existe alguna diferencia entre realizar una división por 0 en un valor entero y en un valor real (*float* o *double*)?

La división por 0 en un valor entero da un error de ejecución: ArithmeticException: /by zero.

La division por 0 en un valor real nos devuelve infinito.

1. Indica el orden en que se ejecutan las siguientes expresiones lógicas e indica el resultado final siendo a=5, b=12 y c=3
   1. (a > b || b >= 12 && b > c)

1. El operador “and” tiene preferencia frente al “or”, por lo tanto, nos centraremos primero en este.

2. Se comparan b y 12: b es igual a 12: true.

3. Se comparan b y c: b es mayor que c: true.

4. Se aplica “and” entre los dos resultados anteriores: uno es true y el otro true, por lo tanto: true.

5. Guardamos ese resultado que se convierte en la parte derecha del operador “or” que analizaremos más tarde.

6. Se comparan a y b: b es mayor que a: false.

7. Se aplica “or” entre los dos resultados anteriores: uno es false y el otro es true, por lo tanto: true.

8. Resultado final: true

* 1. ((a > b || b >= 12) && b > c)

1. Los paréntesis tienen preferencia, así que nos metemos en el paréntesis de más adentro.

2. Se comparan a y b: a es menor que b: false.

3. Como hay un operador “or”, que hará que aun sigua siendo posible obtener true, se hace la siguiente comparación.

4. Se comparan b y 12: b es igual a 12: true.

5. Se aplica “or” entre los dos resultados anteriores: uno es false y el otro true, por lo tanto: true.

6. La solución de los primeros paréntesis es true.

7. Como hasta aquí tenemos true y el siguiente es un operador “and” comprobaremos la comparación que hay a continuación.

8. Se comparan b y c: b es mayor que c: true.

9. Se aplica “and” entre los dos resultados anteriores: uno es true y el otro es true, por lo tanto: true

10.Resultado final: true

* 1. (a > b && b >= 12 || b > c)

1. El operador “and” tiene preferencia frente al “or”, por lo tanto, nos centraremos primero en este.

2. Se comparan a y b: a es menor que b: false. Como está a la izquierda de un operador “and”, no necesitamos hacer la comparación que se encuentra a su derecha. Sea cual sea el resultado, la solución de este operador “and” será false.

3. Guardamos este resultado como la parte que queda a la izquierda del operador “or”.

4. Se comparan b y c: b es mayor que c: true.

5. Se aplica el operador “or” a los dos resultados anteriores: uno es false y el otro true, por lo tanto: true.

6. Resultado final: true.

* 1. ((a > b && b >= 12) || b > c)

Los paréntesis tienen preferencia, así que nos metemos en el paréntesis de más adentro. Al hacer esto, estamos en el mismo caso en cuanto a orden de operaciones que en el apartado anterior. Además, las variables son las mismas, por lo que el resultado también es el mismo: true

1. Si W, X, Y y Z son variables de tipo *boolean* con valores W = false, X = true, Y = true, Z = false, determina el valor de las siguientes expresiones lógicas:
2. Y || !(Y || Z && W) True
3. !X && Y && (!Z || !X) False