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

2. Indica cuál es el tipo primitivo de números reales que permite un mayor rango de números.

   double

3. 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=	byte	short	int	long	char	

4. 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	final int TOTAL=34

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

`float a = 4.6`	El literal 4.6 es de tipo *double* pues es el tipo que toma un literal con decimales si no se pone ningún sufijo. Como un valor *double* tiene un rango mayor que el tipo *float* no se puede realizar la asignación de forma directa.
`byte a = -128`	Es correcto. El literal -128 es realmente un entero por ser el tipo por defecto de los literales, pero se produce una conversión automática cuando el literal está dentro del rango del tipo de la variable a la que se asigna.
`double a = 4.6`	Es correcto. El literal 4.6 es un *double* pues es el tipo que asume cuando no se pone ningún sufijo.
`float a = (float) 4.6`	Es correcto. Al hacer el casting (*float*) se fuerza la conversión del tipo 4.6, que es *double*, en un *float*.
`int d = 45L`	Es incorrecto. No podemos asignar un literal de tipo *long* a una variable de tipo *int*.

`int d = (int) 45L`	Es correcto. ya que convertimos el literal 45L en un entero al hacer (*int*).

a.

6. Indica cuál es la afirmación correcta:
   a. El flujo de datos de impresión se representa con la instancia estática **System.out** de la clase **PrintStream**.
   b. 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**.
   c. El flujo de datos de impresión se representa con la instancia de **System** y su método **out**.
   d. El flujo de datos de impresión se representa con el paquete **System.out**.

La relación entre out, System y PrinStream sería algo como:

```
class System {
   public static PrintStream out;
}
class PrintStream {
    public void println (….){…}
}
```

7. Completa la afirmación siguiente:

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

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

8. 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.

   Import java.util.Scanner

9. 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._____;
if (numero % 2 == 0) {
  System.out.println("el numero " + numero + " es par.");
} else {
   System.out.println("el numero " + numero + " es impar.");
}
```

input.nextInt()

10. 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	

11. 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

12. 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
}
```

```
try {
   //Instrucciones del bloque try
} catch(Exception e) {
   //Instrucciones del bloque catch
} finally {
    instrucciónA;
}
```

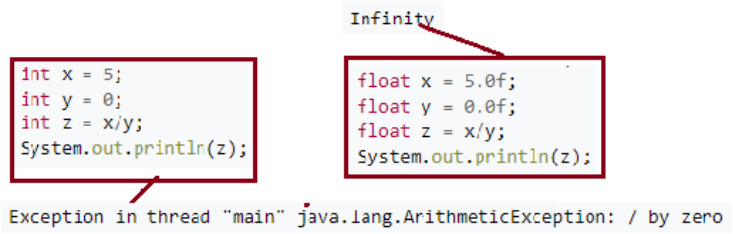
13. Analiza el siguiente código y escríbelo de nuevo para resolver los problemas que puede generar. Explica dónde está el problema y por qué se produce.

```
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");
  }
```

```
 int[] array = new int[20];
 try {
     array[20] = 24;
     int b = 0;
     int a = 23 / b;
  } catch (ArrayIndexOutOfBoundsException excepcion) {
      System.out.println(" Error de índice en un array");
  } catch (ArithmeticException excepcion) {
      System.out.println(" Error aritmético");
  } catch (Exception excepcion) {
      System.out.println(" Error desconocido");
  }
```

Al capturar en el primer catch un objeto *Exception* se impide que se puedan llegar a capturar alguna vez ArithmeticException y ArrayIndexOutOfBoundsException

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



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

    a. `(a > b || b >= 12 && b > c )`
        1. a>b→ false
        2. b>=12→true
        3. b>c→true

        false || true && true

        4. true&&true → true
        5. false || true → true

    b. `((a > b || b >= 12) && b > c )`
        1. a>b→ false
        2. b>=12→true
        3. false || true → true

        true && b>c → true

        4. b>c → true
        5. true && true → true

16. 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:
    a) `Y || !(Y || Z && W)`

        1) Y || !(Y || *false*)

        3) Y || !(true)

        4) Y || false

        5) true

    b) `!X && Y && (!Z || !X)`
        1. !X && Y && (true|| !X)
        2. !X && Y && (true|| false)
        3. !X && Y && (true)
        4. false && Y && (true)
        5. false && true
        6. false