Capitulo 1
Certificación de Java 17

Building blocks

1.	D,E. En el método main: • las características public, static y void, son indispensables. • final no es necesario, pero no afectaría que estuviese. • args podría tener cualquier otro nombre y continuaría funcionando.
2.	C,D,E. El orden de los elementos es siempre el siguiente: 1. package 2. import 3. class En caso de que faltara algún elemento, el orden igualmente debería mantenerse.
3.	A,E. El nombre de la clase de establece después de class y, por buenas prácticas, con UpperCamellCase. Por lo que sabemos que Bunny es la clase. Para establecer una variable de referencia, se establece primero de qué tipo es, por ejemplo, de tipo String, StringBuilder, o en este caso, de tipo Bunny. Inmediatamente después, el nombre del valor de referencia, un = que indica a dónde apunta. Por lo que sabemos que bun es una referencia al objeto.
4.	B,E,G. Los identifiers deben seguir las siguientes reglas: 1. Iniciar con una letra, símbolo $ o guión bajo _ 2. Pueden incluir letras, dígitos, símbolos de moneda o guiones bajos después del primer caracter. 3. No palabras reservadas (dado que Public fue escrito con mayúscula, es válido) 4. No true, false o null 5. Un sólo guión bajo no está permitido. 6. No pueden iniciar con número.
5.	A,D,F.

	Llamar al garbage collector con System.gc(); no garantiza que se llevará todas las variables sin referencia. Incluso no es necesario llamarlo, éste igual vendrá en el momento que decida a llevarse los objetos que queden sin referencia. Al establecer null, la referencia que apunta hacia un objeto desaparece, sin embargo, si alguna otra variable de referencia quedó apuntando a éste, no puede ser llevado por el gc, hasta que todas las conexiones desaparezcan y no haya nadie
6.	F. Las variables que se encuentran dentro de bloques de inicialización sólo viven dentro del bloque, por lo que no serían accesibles. Las variables que sí podrían imprimirse serían: water, air, twoHumps, distance, path, age, i.
7.	NO ES NECESARIO RESOLVER.
8.	B,D,E,H. Cuando usamos var, no es posible inicializarlo con null porque no se podría inferir de qué tipo es (A es incorrecta), pero sí podría establecerse como null más adelante, en tanto no sea primitivo. Es por ello que H es correcta, pues inició como String y después se estableció como null. C no es posible porque como es int no puede pasar a null, podría más bien pasar a 0. E sí compila, no arroja error de compilación, en tiempo de compilación todo es correcto, pero en tiempo de ejecución, sí arrojará error. F y G no son posibles ya que var no permite asignación de múltiples variables. Var solo puede definirse dentro de un método.
9.	E Defaults: • en el caso de objetos (incluyendo String o arrays): null • en el caso de boolean: false • en el caso de float: 0.0 • en el caso de char : no muestra nada • en el caso de short, byte, int: 0 • en el caso de long: ? 0L Nota: Estos valores por defecto únicamente se aplican a:

- Variables de instancia (campos de clase).
- Variables estáticas.

Las variables locales NO tienen valores por defecto:

- Deben ser inicializadas explícitamente antes de usarlas.
- El compilador mostrará un error si intentas usar una variable local no inicializada.

10.

A,E,F.

El guión bajo no se permite al inicio, al final o junto a un punto decimal.

11.

E

Java.lang se importa de manera automática, por default, en Java por lo que no es necesario hacer import del mismo. Y ya que ambas clases (Water y Tank) están en el mismo paquete, no son necesarios los import 3 y 4. Por lo que se podrían borrar todos los import y seguir compilando.

12.

A,C,D.

- Línea 2: necesitaríamos cambiar la coma por punto y coma para separar datos de diferente tipo o generaría error de compilación. Si fueran del mismo tipo, sí se podría.
- Línea 3: es un bloque de inicialización, inicia la variable en cero.
- Línea 4: no admite establecer valores predeterminados para los parámetros de los métodos.

```
// Esto NO es válido en Java
public void ejemplo(String nombre = "John", int edad = 30) {
    // código
}
```

- Se necesitarían crear múltiples métodos sobrecargados:

```java
// La forma correcta en Java usando sobrecarga de métodos
public void ejemplo() {
    ejemplo("John", 30);  // Valores predeterminados
}

public void ejemplo(String nombre) {
    ejemplo(nombre, 30);  // Solo edad predeterminada
```

13.

A,B,C

import.aquarium.Water llama específicamente a esa clase , por lo que incluso si tuviera también import.jellies.* compilaría.

Cuando no se define si es public o private, por default solo será private, por lo que no será accesible.

El import que esté más explícito es el que se tomará primero.

Si los dos import son generales con *, y en ambos casos tengo una variable con el mismo nombre, la IDE no sabría cuál tomar, por lo que no compilaría, a menos que se defina todo el camino al usarse.

14.

A,B,D,E

- Línea 3: genera error por marcar la L de long dentro de un short. Un long NO cabe en un short.
- Línea 4: genera un error porque intentamos meter un double dentro de un int.
- Línea 6: marca error porque, incluso si estuviese correctamente declarada la variable, es un primitivo y no se les puede aplicar comportamientos o métodos, los métodos sólo serían de los objetos.
- Línea 7: de la misma manera no se le puede asignar comportamiento.
- Línea 8: el comportamiento se aplica a un String que sí es un objeto, por lo que sí es posible asignarle método y es correcta.

15.

C,E,F

- A. llamar explícitamente al gc NO es garantía de que se lleve los objetos.
- B. no tiene un tiempo determinado en que aparezca.
- C. el gc permite a la JVM obtener espacio de memoria al eliminar lo que no es necesario.
- D. no se tiene certeza de cuándo corre el gc.
- E. “An object may be elegible for the garbage collector but never removed from the heap.” Esto es posible, podría ser elegido, pero no llevárselo.
- F. los objetos sin referencia son elegibles.
- G. no hay relación entre el gc y el final.

16.	NO ES NECESARIO RESOLVER.
17.	D,F, Las variables de instancia (línea 2 y 3), así como las static (línea 4) tienen un default. • String = null • boolean = false • float = 0.0 • La línea 7 imprimiría Empty = false • La línea 8 imprimiría Brand = null • La línea 9, no necesita la variable de referencia porque, toda vez que es static, le pertenece a la clase (si pusieramos el objeto lanzaría advertencia pero no error). De tal manera que, es correcto y arrojaría Code = 0.0
18.	B,C,F • A. var NO se puede usar como parámetro de constructor. Principal (var name){} //marcaría error • B. el tipo de var siempre debe ser conocido en tiempo de compilación, si no se puede inferir el tipo de var, lanzará error. • C. var no se puede usar como variable de instancia o static, **únicamente usamos var para variables locales.** • D. var no se puede usar para la asignación de múltiples variables. • E. el valor sí se puede cambiar, el tipo no. • F. el tipo nunca se va a poder cambiar. Si es String, siempre será String • G. var NO es una palabra reservada de Java.
19.	A,D. • var num1 = Long.parseLong (“100”); parseLong(String s) recibe un String y lo parsea a un long primitivo. • var num2 = Long.valueOf(“100”); valueOf(String s) nos da su wrapper Long. • System.out.println(Long.max(num1, num2)); la clase max(long a, long b) solamente recibe dos primitivos. num1 es primitivo, pero num2 es un wrapper. Al usar el método .max(), Java aplicará un unboxing

	(transformará el wrapper en primitivo). num1 es un primitivo, num2 no lo es porque es un Long (el wrapper de long).
20.	C. En la línea 14, public void PoliceBox(){} sabemos que esto no es un constructor, pues los constructores no tienen tipo de retorno, es decir void. Por tanto, los valores se inicializan con sus default y no con los valores en las líneas 5 y 6. En las líneas 11 y 12, cambio a donde apunta p, sin embargo en la línea 13, le digo a p que apunte a donde apunta q, quien sigue apuntando a los valores default. Por tanto: • q.color=null • q.age = 0.0 • p.color=null • p.color=null
21.	D 1. Primero entramos por el método main, en donde lo primero que hace es imprimir el 7- 2. En main sí se puede usar var porque está dentro de un método. var crea new Salmon. 3. Al crear new Salmon, tendríamos primero que ejecutar los bloques static en el orden que vayan apareciendo, pues lo static se ejecuta primero, pero como no hay, seguimos con bloques de instancia. 4. El primer bloque **{ System.out.print(count+"-"); }** imprime el valor de count que se inicializó con default como **0** más un guión **-** 5. El siguiente bloque **{ count++; }** incrementa count en uno pero no lo imprime 6. Ahora pasamos al constructor, en donde a count se le asigna valor de 4 y manda a imprimir **2-** 7. Ahora sí terminamos de crear new Salmon, y podemos pasar al último System.out en main **System.out.print(s.count+"-").** Imprime el valor de count que es **4** más el guión **-** **//7-0-2-4-**
22.	C,F,G. A. es un binario que bien puede estar en int, pero el nombre de la variable inicia con mayúscula, y en print, la variable está con minúscula, por lo que nunca la encontraría. B. no podemos meter un long en un int por memoria.

	C. es un hexadecimal que sí puede estar en un int, así que sí compila. D. sería un dato double que no se puede meter en un int. E. el guión no puede ir junto al punto. F. es un binario porque inicia con 0B o 0b, también entra en un int y es correcto. G. es un double y es correcto .
23.	A,D. A. float temp = 50.0; el 50.0 es un double porque no tiene F de float, pero el double es más grande que float, por tanto no cabe, marcaría error. Tendríamos que cambiar el tipo a double o agregar la f de float. B. la línea 6 no genera error. C. la línea 7 no genera error D. el alcance de depth sólo está dentro del for, por lo que no lo alcanza a leer desde fuera. E, F,G,H no imprimiría nada por los errores.