1 ¿Qué arroja?

```java
public class Main {
    public static void main(String[] args) {
        String[] at = {"FINN", "JAKE"};
        for (int x = 1; x < 4; x++) {
            for (String s : at) {
                System.out.println(x + " " + s);
                if (x == 1) {
                    break;
                }
            }
        }
    }
}
// Salida:
// 1 FINN
// 2 FINN
// 2 JAKE
// 3 FINN
// 3 JAKE
```

Explicación:
La primera iteración del primer ciclo for es detenida debido a la condición if, pero las siguientes iteraciones se completan con normalidad.

2 ¿Qué 5 líneas son correctas?

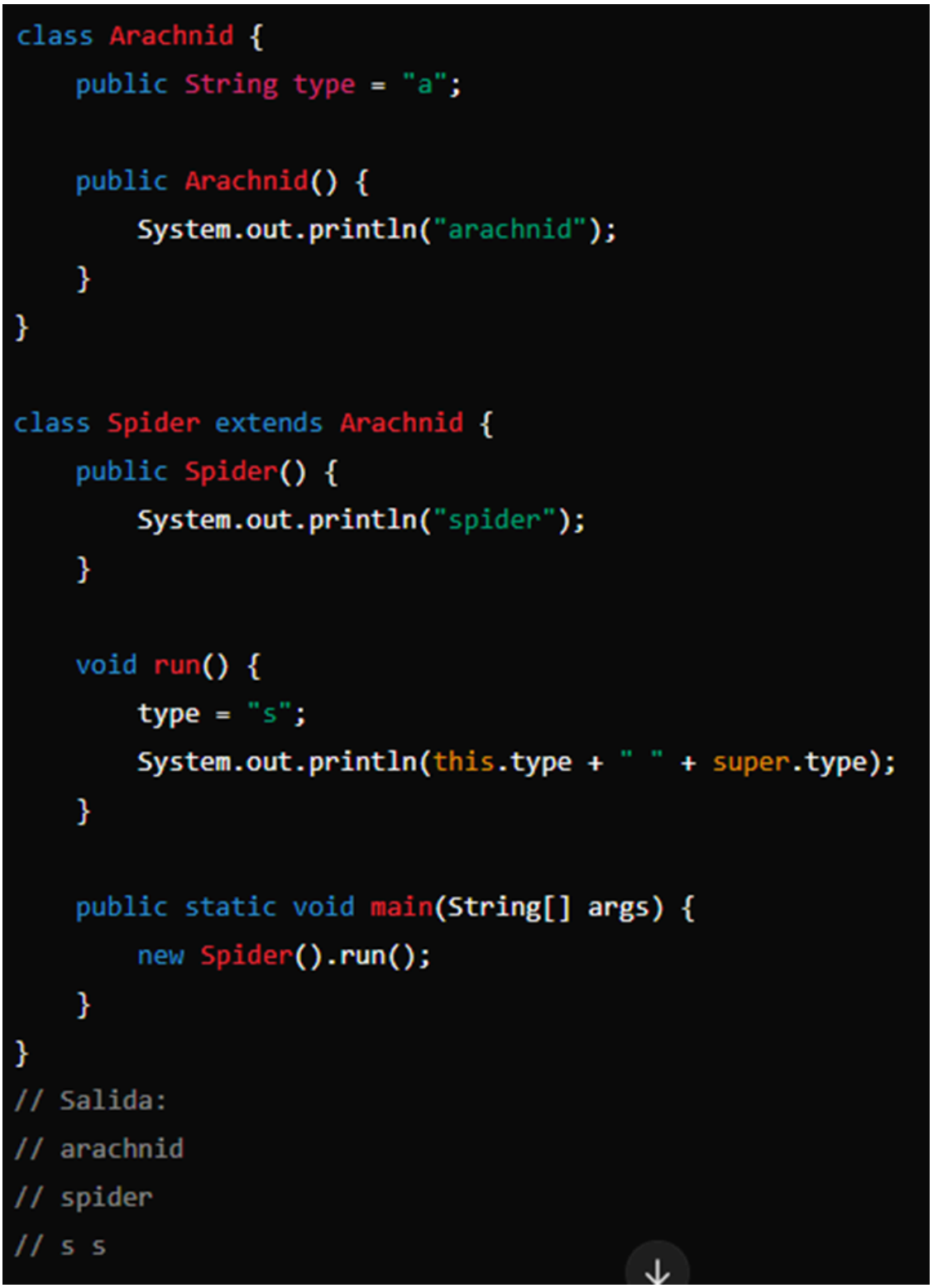
```
class Light {
    protected int lightsaber(int x) { return 0; }
}

class Saber extends Light {
    // Línea incorrecta: private no puede ser más restrictivo que protected
    // private int lightsaber(int x) { return 0; }

    // Sobrecarga válida (método diferente)
    protected int lightsaber(long x) { return 0; }

    // Sobrecarga válida (método diferente)
    private int lightsaber(long x) { return 0; }

    // Error: firma del método cambia el tipo de retorno, no es sobrescritura válida
    // protected long lightsaber(int x) { return 0; }

    // Sobrecarga válida (método diferente)
    protected long lightsaber(int x, int y) { return 0; }

    // Sobrescritura válida (el modificador de acceso es más amplio o igual)
    public int lightsaber(int x) { return 0; }

    // Sobrecarga válida (método diferente)
    protected long lightsaber(long x) { return 0; }
}
```

3 ¿Qué resultado arroja?

```java
class Mouse {
    public int numTeeth;
    public int numWhiskers;
    public int weight;

    public Mouse(int weight) {
        this(weight, 16);
    }

    public Mouse(int weight, int numTeeth) {
        this(weight, numTeeth, 6);
    }

    public Mouse(int weight, int numTeeth, int numWhiskers) {
        this.weight = weight;
        this.numTeeth = numTeeth;
        this.numWhiskers = numWhiskers;
    }

    public void print() {
        System.out.println(weight + " " + numTeeth + " " + numWhiskers);
    }

    public static void main(String[] args) {
        Mouse mouse = new Mouse(15);
        mouse.print();
    }
}
// Salida: 15 16 6
```

Explicación:
Al instanciar el objeto con el constructor que recibe un argumento de tipo int, va pasando por los siguientes constructores que se van llamando dentro de cada constructor, definiendo finalmente todas las variables del método print.

4 ¿Cuál es la salida?



```java
class Arachnid {
    public String type = "a";

    public Arachnid() {
        System.out.println("arachnid");
    }
}

class Spider extends Arachnid {
    public Spider() {
        System.out.println("spider");
    }

    void run() {
        type = "s";
        System.out.println(this.type + " " + super.type);
    }

    public static void main(String[] args) {
        new Spider().run();
    }
}
// Salida:
// arachnid
// spider
// s s
```

Al crear una instancia de un objeto de tipo `Spider`, primero se ejecuta el constructor de su superclase, seguido por el constructor de la propia subclase. Posteriormente, en el método `run`, se modifica el valor del atributo `type`. Dado que tanto la superclase como la subclase comparten el mismo atributo, `this.type` y `super.type` se refieren al mismo atributo en memoria.

5 Resultado:

```
class Test {
    public static void main(String[] args) {
        int b = 4;
        b--;
        System.out.println(--b);
        System.out.println(b);
    }
}

class Sheep {
    public static void main(String[] args) {
        int ov = 999;
        ov--;
        System.out.println(--ov);
        System.out.println(ov);
    }
}
// Salida para Sheep:
// 997
// 997
```

La salida es la misma debido a que el operador de pre-decremento actúa antes de la salida a la consola.

6 resultado:

```java
class Overloading {
    public static void main(String[] args) {
        System.out.println(overload("a"));         // Llama a overload(String s)
        System.out.println(overload("a", "b"));    // Llama a overload(String s, String t)
        System.out.println(overload("a", "b", "c")); // Llama a overload(String... s)
    }

    public static String overload(String s) {
        return "1";
    }

    public static String overload(String... s) {
        return "2";
    }

    public static String overload(Object o) {
        return "3";
    }

    public static String overload(String s, String t) {
        return "4";
    }
}
// Salida:
// 1
// 4
// 2
```

La Salida es en ese orden por los parámetros especificados en cada llamada a los métodos sobreescritos.

7 Resultado:

```java
class Base {
    public void test() {
        System.out.println("Base");
    }
}

class Base1 extends Base {
    public void test() {
        System.out.println("Base1");
    }
}

class Base2 extends Base {
    public void test() {
        System.out.println("Base2");
    }
}

class Test {
    public static void main(String[] args) {
        Base obj = new Base1();
        ((Base2) obj).test();  // Genera ClassCastException
    }
}
// Resultado: ClassCastException
```

Esto genera la Excepción ClassCastException en tiempo de ejecución. Debido a que un objeto de Clase Base1 no puede ser referenciado a una variable de referencia Base2.

8 Resultado:

```java
public class Fish {
    public static void main(String[] args) {
        int numFish = 4;
        String fishType = "Tuna";
        String anotherFish = numFish + 1;
        System.out.println(anotherFish + " " + fishType);
        System.out.println(numFish + " " + 1);
    }
}
// Resultado: El código no compila porque `anotherFish` debería ser una String concatenada
```

// Resultado: El código no compila porque `anotherFish` debería ser una String concatenada, no int + int

9 Resultado

```java
class MathFun {
    public static void main(String[] args) {
        int number1 = 0b0111;   // Número binario
        int number2 = 0111_000; // Número octal

        System.out.println("Number1: " + number1);
        System.out.println("Number2: " + number1);  // El valor se imprime dos veces
    }
}
// Salida:
// Number1: 7
// Number2: 7
```

10 Resultado:

```java
class Calculator {
    int num = 100;

    public void calc(int num) {
        this.num = num * 10;
    }

    public void printNum() {
        System.out.println(num);
    }

    public static void main(String[] args) {
        Calculator obj = new Calculator();
        obj.calc(2);
        obj.printNum();
    }
}
// Salida: 20
```

el método calc si actúa sobre el argumento de esa instancia debido a la palabra reservada .this

11 ¿Qué aseveraciones son correctas?

```java
import java.util.Random;

class ImportExample {
    public static void main(String[] args) {
        Random r = new Random();
        System.out.println(r.nextInt(10));
    }
}
// • If you omit java.util import statements, java compiles gives you an error
// • java.lang and util.random are redundant
// • you don't need to import java.lang
```

12 Resultado:

```java
public class Main {
    public static void main(String[] args) {
        int var = 10;
        System.out.println(var++);
        System.out.println(++var);
    }
}
// Salida:
// 10
// 12
```

13 Resultado

```java
class MyTime {
    public static void main(String[] args) {
        short mn = 11;
        short hr;
        short sg = 0;
        for (hr = mn; hr > 6; hr -= 1) {
            sg++;
        }
        System.out.println("sg=" + sg);
    }
}
// Salida: sg=5
```

Cuando el valor de hr llega a 6, se interrumpe el ciclo y sg tiene el valor de 5.

14 ¿Cuáles son verdad?

```java
// • An ArrayList is mutable
// • An Array has a fixed size
// • An array is mutable
// • An array allows multiple dimensions
// • An ArrayList is ordered
// • An array is ordered
```

15 Resultado:

```
public class MultiverseLoop {
    public static void main(String[] args) {
        int negotiate = 9;
        do {
            System.out.println(negotiate);
        } while (--negotiate);  // Error: necesita una expresión booleana en el `while`
    }
}
// Error de compilación
```

16 Resultado:

```
import java.util.stream.Stream;

class App {
    public static void main(String[] args) {
        Stream<Integer> nums = Stream.of(1, 2, 3, 4, 5);
        nums.filter(n -> n % 2 == 1);
        nums.forEach(p -> System.out.println(p));
    }
}
// Resultado: Exception at runtime, se debe encadenar el stream porque se consume
```

El problema radica en que el stream `nums` no se encadenó correctamente. Cuando llamas a `filter`, se crea un nuevo stream, pero ese stream no se almacena ni se utiliza. Luego, al intentar usar el stream original `nums` en `forEach`, este sigue siendo el stream sin filtrar. Sin embargo, dado que el stream original ya fue utilizado (o está marcado para ser consumido), el código intenta procesar un stream que ya ha sido consumido, lo que provoca una excepción en tiempo de ejecución (`IllegalStateException`).

Solución correcta:

Para evitar esta excepción, debes encadenar las operaciones del stream:

java Copiar cód

```
import java.util.stream.Stream;

class App {
    public static void main(String[] args) {
        Stream<Integer> nums = Stream.of(1, 2, 3, 4, 5);
        nums.filter(n -> n % 2 == 1)
            .forEach(p -> System.out.println(p));  // Encadenar el filter y forEach
    }
}
```

Salida esperada con la corrección:

Copiar código

```
1
3
5
```

17 Pregunta
suppose the declared type of x is a class, and the declared type of y is an interface. When is the assignment x = y; legal?

• When the type of X is Object

18 Pregunta
when a byte is added to a char, what is the type of the result?
• int

19 Pregunta
the standard application programming interface for accessing databases in java?

- JDBC segun CHATGPT

20 Pregunta
Which one of the following statements is true about using packages to organize your code in Java ?
• Packages allow you to limit access to classes, methods, or data from classes outside the package.

21 Pregunta
Forma correcta de inicializar un booleano:
• boolean a = (3>6);

Pregunta 23.

```
class Y {
    public static void main(String[] args) throws IOException {
        try {
            doSomething();
        } catch (RuntimeException exception) {
            System.out.println(exception);
        }
    }

    static void doSomething() throws IOException {
        if (Math.random() > 0.5) {
            // Do nothing
        }
        throw new RuntimeException();
    }
}
```

El código intenta ejecutar un método (`doSomething()`) que puede lanzar una excepción. Dentro del método, si una condición aleatoria no se cumple, se lanza una

`RuntimeException`. Esta excepción es capturada por el bloque `catch`, que la imprime en la consola. Así que el resultado del programa será que se imprime la excepción `java.lang.RuntimeException`.

Pregunta 24

```
interface Interviewer {
    abstract int interviewConducted();
}

public class Manager implements Interviewer {
    int interviewConducted() {
        return 0;
    }
}
```

En la interfaz `Interviewer`, el método `interviewConducted()` es abstracto y, por lo tanto, implícitamente `public`. Sin embargo, en la clase `Manager`, el método `interviewConducted()` está declarado con la visibilidad por defecto (package-private), lo cual es menos restrictivo que `public`. Java no permite reducir la visibilidad al sobrescribir un método, por lo que el compilador arrojará un error indicando que el método en `Manager` debe ser `public` para que sea válido como implementación del método de la interfaz `Interviewer`.

Pregunta 25

```
class Arthropod {
    public void printName(double input) {
        System.out.println("Arth");
    }
}

class Spider extends Arthropod {
    public void printName(int input) {
        System.out.println("Spider");
    }

    public static void main(String[] args) {
        Spider spider = new Spider();
        spider.printName(4);   // Llamará al método de Spider
        spider.printName(9.0); // Llamará al método de Arthropod
    }
}
```

Básicamente en el código están ocupando Overloading de métodos.
spider.printName(4) llama al método printName(int) en Spider → Imprime "Spider".
spider.printName(9.0) llama al método printName(double) en Arthropod → Imprime "Arth".

Pregunta 26

```
public class Main {
    public enum Days {Mon, Tue, Wed}

    public static void main(String[] args) {
        for (Days d : Days.values()) {
            Days[] d2 = Days.values();
            System.out.println(d2[2]);
        }
    }
}
```

El código imprime "Wed" tres veces.

1. **Enum Days**: Define tres valores: `Mon`, `Tue`, `Wed`.
2. **`for` loop**: Itera sobre los valores del enum, pero dentro del bucle se crea un nuevo array `Days[]` d2 en cada iteración.
3. **`d2[2]`**: Siempre se refiere a `Wed`, que es el tercer valor del enum.

Pregunta 27

```
public class Main {
    public enum Days {MON, TUE, WED};

    public static void main(String[] args) {
        boolean x = true, z = true;
        int y = 20;
        x = (y != 10) ^ (z = false);
        System.out.println(x + " " + y + " " + z);
    }
}
```

Respuesta: true 20 false

El código evalúa una expresión lógica utilizando el operador XOR (^). La variable x se asigna el resultado de `true ^ false`, que es `true`, mientras que la variable z se asigna a `false`. El valor de y no se modifica. Por eso, el resultado impreso es `true 20 false`.

Pregunta 28

```
class InitializationOrder {
    static { add(2); }

    static void add(int num) {
        System.out.println(num + "");
    }

    InitializationOrder() {
        add(5);
    }

    static { add(4); }

    { add(6); }

    static { new InitializationOrder(); }

    { add(8); }

    public static void main(String[] args) {}
}
```

Respuesta: 2 4 6 8 5

el recorrido empieza con los bloques static, seguido de los bloques de inicialización (6 y 8) y finalmente pasa por el constructor (5)

Pregunta 29

```
public class Main {
    public static void main(String[] args) {
        String message1 = "Wham bam";
        String message2 = new String("Wham bam");
        if (message1 != message2) {
            System.out.println("They don't match");
        } else {
            System.out.println("They match");
        }
    }
}
```

Respuesta: They dont match

Explicación: Los strings son diferentes porque el string en la variable `message2` no se almacena en el pool de strings. En su lugar, apunta a un objeto en una ubicación distinta en la memoria. Por lo tanto, la comparación de referencias resulta en `false`.

Pregunta 30

```
class Mouse {
    public String name;

    public void run() {
        System.out.println("1");
        try {
            System.out.println("2");
            name.toString();
            System.out.println("3");
        } catch (NullPointerException e) {
            System.out.println("4");
            throw e;
        }
        System.out.println("5");
    }

    public static void main(String[] args) {
        Mouse jerry = new Mouse();
        jerry.run();
        System.out.println("6");
    }
}
```

Salida: 1 2 4 NullPointerException

El problema radica en que la llamada a `name.toString()` en un objeto `null` lanza una `NullPointerException`, que se captura en el bloque `catch`. La excepción se vuelve a lanzar dentro del `catch`, interrumpiendo la ejecución del método y propagando el error. Como resultado, el programa termina abruptamente porque la excepción no se maneja en `main`, lo que evita que el código posterior se ejecute.

Pregunta 31

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

public class Main {
    public static void main(String[] args) {
        try (Connection con = DriverManager.getConnection(url, uname, pwd)) {
            Statement stmt = con.createStatement();
            System.out.print(stmt.executeUpdate("INSERT INTO User VALUES (500, 'Ramesh')"));
        }
    }
}
```

Salida: arroja 1

Este código ilustra cómo se ejecuta una inserción en una base de datos desde un programa Java utilizando JDBC. La salida del programa es `1`, que representa el número de filas afectadas por la operación `INSERT`.

Pregunta 32

```
class MarvelClass {
    public static void main(String[] args) {
        MarvelClass ab1, ab2, ab3;
        ab1 = new MarvelClass();
        ab2 = new MarvelMovieA();
        ab3 = new MarvelMovieB();
        System.out.println("The profits are " + ab1.getHash() + ", " + ab2.getHash() + ",
    }

    public int getHash() {
        return 676000;
    }
}

class MarvelMovieA extends MarvelClass {
    public int getHash() {
        return 18330000;
    }
}

class MarvelMovieB extends MarvelClass {
    public int getHash() {
        return 27980000;
    }
}
```

La salida del código muestra `the profits are 676000,18330000,27980000` porque cada instancia de `MarvelClass` y sus subclases llama a su respectivo método `getHash`. `ab1`, una instancia de `MarvelClass`, retorna `676000` usando su propia implementación del método. `ab2`, una instancia de `MarvelMovieA`, sobrescribe `getHash` y retorna `18330000`. `ab3`, una instancia de `MarvelMovieB`, también sobrescribe `getHash` y retorna `27980000`. La concatenación de estos valores en la llamada a `System.out.println` produce la salida final.

Pregunta 33

```
class Song {
    public static void main(String[] args) {
        String[] arr = {"DU HAST", "FEEL", "YELLOW", "FIX YOU"};
        for (int i = 0; i <= arr.length; i++) {
            System.out.println(arr[i]);
        }
    }
}
```

Respuesta: // Este código lanzará una ArrayIndexOutOfBoundsException.

El arreglo `arr` tiene 4 elementos, por lo que los índices válidos son `0`, 1, 2, y 3. Sin embargo, el bucle se ejecuta hasta `i <= arr.length`, lo que significa que en la última

iteración, `i` tomará el valor 4, intentando acceder a `arr[4]`, que no existe, lo que genera la excepción.

Pregunta 34

```
class Menu {
    public static void main(String[] args) {
        String[] breakfast = {"beans", "egg", "ham", "juice"};
        for (String rs : breakfast) {
            int dish = 2;
            while (dish < breakfast.length) {
                System.out.println(rs + "," + dish);
                dish++;
            }
        }
    }
}
```

El código produce un total de **8 líneas** de salida (2 por cada uno de los 4 elementos en el arreglo `breakfast`).

Pregunta 35

```
// Which of the following statements are true:

// • StringBuilder is generally faster than StringBuffer.
// • StringBuffer is threadsafe; StringBuilder is not.
```

Pregunta 36

```
class CustomKeys {
    Integer key;

    CustomKeys(Integer k) {
        key = k;
    }

    public boolean equals(Object o) {
        return ((CustomKeys) o).key == this.key;
    }
}
```

Este código producirá un error de compilación porque en el método `equals`, se está utilizando el operador == para comparar objetos de tipo `Integer`. En lugar de ==, se debe utilizar el método `equals` para comparar los valores de los objetos `Integer`.

Pregunta 37

```
// The catch clause is of the type:

// - Throwable
// - Exception but NOT including RuntimeException
// - CheckedException
// - RuntimeException
// - Error
```

Pregunta 38

```
// An enhanced for loop
// • Also called "for each", offers simple syntax to iterate through a collection
//   but it can't be used to delete elements of a collection.
```

Pregunta 39

```
// Which of the following methods may appear in class Y, which extends X?

// public void doSomething(int a, int b) { … }
```

Pregunta 40

```
public class Main {
    public static void main(String[] args) {
        String s1 = "Java";
        String s2 = "java";
        if (s1.equalsIgnoreCase(s2)) {
            System.out.println("Equal");
        } else {
            System.out.println("Not equal");
        }
    }
}
```

En este código, se utiliza el método `equalsIgnoreCase` para comparar las cadenas `s1` y `s2`. Este método compara las dos cadenas sin tener en cuenta las mayúsculas y minúsculas, por lo que el resultado es "Equal" al ejecutarlo.

Pregunta 41

```java
import java.util.Arrays;

class App {
    public static void main(String[] args) {
        String[] fruits = {"banana", "apple", "pears", "grapes"};

        // Ordenar el arreglo de frutas utilizando compareTo
        Arrays.sort(fruits, (a, b) -> a.compareTo(b));

        // Imprimir el arreglo de frutas ordenado
        for (String s : fruits) {
            System.out.println(s);
        }
    }
}

/*
Salida esperada:
apple
banana
grapes
pears
*/
```

En este código, el método `Arrays.sort` se utiliza con una expresión lambda `(a, b) -> a.compareTo(b)` para ordenar el arreglo `fruits` en orden alfabético. El método `compareTo` compara dos cadenas lexicográficamente, lo que permite que el arreglo se ordene de la siguiente manera: apple, banana, grapes y pears.

Pregunta 42

```java
public class Main {
    public static void main(String[] args) {
        int[] countsofMoose = new int[3];
        System.out.println(countsofMoose[-1]);
    }
}

// Este código lanzará una ArrayIndexOutOfBoundsException
```

Intentar acceder a un índice negativo (o fuera de los límites del arreglo) generará una excepción `ArrayIndexOutOfBoundsException`.

Pregunta 43

```java
class Salmon {
    int count;

    public void Salmon() {
        count = 4;
    }

    public static void main(String[] args) {
        Salmon s = new Salmon();
        System.out.println(s.count);
    }
}
```

Cuando se crea un objeto `Salmon s = new Salmon();`, el campo `count` no se inicializa a 4 porque el método `Salmon()` nunca es llamado. En su lugar, `count` conserva su valor predeterminado de `0` para los campos `int`.

44 Pregunta

```java
class Circuit {
    public static void main(String[] args) {
        runlap();
        int c1 = c2;  // Error: c2 no está declarado aún
        int c2 = v;   // c2 se declara aquí, pero es demasiado tarde
    }

    static void runlap() {
        System.out.println(v);  // v está declarado pero no inicializado
    }

    static int v;
}
```

Problema: La variable `c2` se usa antes de ser declarada.
Respuesta: // corregir linea 6; c1 se le asigna c2 pero c2 aun no se declara

45 Pregunta

```java
class Foo {
    public static void main(String[] args) {
        int a = 10;
        long b = 20;
        short c = 30;
        System.out.println(++a + b++ * c);
    }
}
```

Problema: El orden de las operaciones y la prioridad de los operadores afectan el resultado.
Salida esperada: `611` (11 + 20 * 30).

46 Pregunta

```java
public class Shop {
    public static void main(String[] args) {
        new Shop().go("welcome", 1);
        new Shop().go("welcome", "to", 2);
    }

    public void go(String... y, int x) {
        System.out.print(y[y.length - 1] + "");
    }
}

// Compilation fails
```

Cuando se utilizan varargs (`String... y`), deben ser el último parámetro en la lista de parámetros del método. Sin embargo, en este caso, `int x` está después del varargs, lo que no es permitido en Java.

47 pregunta

```java
class Plant {
    Plant() {
        System.out.println("plant");
    }
}

class Tree extends Plant {
    Tree(String type) {
        System.out.println(type);
    }
}

class Forest extends Tree {
    Forest() {
        super("leaves"); // Llama al constructor de Tree
        new Tree("leaves"); // Crea una nueva instancia de Tree
    }

    public static void main(String[] args) {
        new Forest();
    }
}
```

Problema: Las clases deben llamar al constructor de la superclase antes de realizar cualquier otra acción.
Salida Esperada:

Copiar código

```
plant
leaves
plant
leaves
```

48 Pregunta

```java
class Test {
    public static void main(String[] args) {
        String s1 = "hello";
        String s2 = new String("hello");
        s2 = s2.intern(); // Interna la cadena, devolviendo la referencia en el pool
        System.out.println(s1 == s2); // Comparar referencias
    }
}
```

El método `intern()` devuelve una referencia a la cadena en el pool de cadenas. En otras palabras, mete a ese objeto en el pool de strings.
Salida esperada: `true`

49 Pregunta: ¿Cuál de las siguientes construcciones es un ciclo infinito while?

```java
// Ambos ciclos son infinitos
while(true);
while(1 == 1) {}
```

Ambos ciclos son infinitos.

50 Pregunta

```java
public class Main {
    public static void main(String[] args) {
        int a = 10;
        int b = 37;
        int z = 0;
        int w = 0;
        if (a == b) {
            z = 3;
        } else if (a > b) {
            z = 6;
        }
        w = 10 * z;
        System.out.println(z); // `z` es 0
    }
}
```

Problema: Ninguna de las condiciones en el `if` se cumple, por lo tanto `z` permanece en 0.
Salida esperada: `0`

51 Pregunta

```java
public class Main {
    public static void main(String[] args) {
        course c = new course();
        c.name = "java";
        System.out.println(c.name);
    }
}

class course {
    String name;
    course() {
        course c = new course(); // Creación recursiva
        c.name = "Oracle";
    }
}
```

Problema: El constructor está creando una nueva instancia recursivamente, lo que lleva a un desbordamiento de pila (`StackOverflowError`).

52 Pregunta

```java
public class Main {
    public static void main(String[] args) {
        String a;
        System.out.println(a.toString());
    }
}
```

Problema: La variable `a` no está inicializada antes de llamar a `toString()`. En Java, las variables locales deben ser inicializadas antes de ser utilizadas.
Respuesta: //builder fails

53 Pregunta

```java
public class Main {
    public static void main(String[] args) {
        System.out.println(2 + 3 + 5); // Suma de números
        System.out.println("+" + 2 + 3 + 5); // Concatenación de cadenas
    }
}
```

Problema: La concatenación de cadenas con números se maneja de manera diferente a la suma de números. **Salida esperada: 10 +235**

54 Pregunta

```java
public class Main {
    public static void main(String[] args) {
        int a = 2;
        int b = 2;
        if (a == b)
            System.out.println("Here1");
        if (a != b)
            System.out.println("here2");
        if (a >= b)
            System.out.println("Here3");
    }
}
```

Son comparaciones simples y su resultado.
Salida Esperada:

```
Here1
Here3
```

55 Pregunta

```java
public class Main extends count {
    public static void main(String[] args) {
        int a = 7;
        System.out.println(count(a, 6));  // Error: no se puede llamar a un método estáti
    }
}

class count {
    int count(int x, int y) {
        return x + y;
    }
}
```

Respuesta: //Build fails

Este código fallará en la compilación por dos razones principales:

1. **Llamada al método no estático desde un contexto estático**: En `main`, se intenta llamar al método `count` de la clase `count` directamente usando `count(a, 6)`. Sin embargo, `count(int x, int y)` no es un método estático; es un método de instancia. Los métodos de instancia deben ser llamados en el contexto de una instancia de la clase.
2. **Método con el mismo nombre que la clase**: El nombre de la clase `count` es el mismo que el nombre del método `count`, lo que puede llevar a confusión. Aunque

no es un error por sí mismo, es una práctica desaconsejada porque puede hacer que el código sea menos claro.

56 Pregunta

```
class trips {
    void main() {
        System.out.println("Mountain");
    }
    static void main(String args[]) {
        System.out.println("BEACH");
    }
    public static void main(String[] args) {
        System.out.println("magic town");
    }
    void mina(Object[] args) {
        System.out.println("city");
    }
}
```

Explicación: Diferentes métodos `main` están definidos, pero solo uno se ejecuta en Java.
Salida esperada: `magic town`

57 Pregunta

```
public class Main {
    public static void main(String[] args) {
        int a = 0;
        System.out.println(a++ + 2); // Imprime 2
        System.out.println(a); // Imprime 1
    }
}
```

Problema: El operador `++` se usa después de la variable, lo que hace que `a` se incremente después de su uso.
Salida esperada: 2, 1

58 Pregunta

```java
import java.util.ArrayList;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        List<E> p = new ArrayList<>();
        p.add(2);
        p.add(1);
        p.add(7);
        p.add(4);
    }
}
```

Este código fallará en la compilación debido a que `E` no está definido. En Java, `E` es típicamente utilizado como un parámetro de tipo genérico en una definición de clase o interfaz, pero aquí se está utilizando directamente sin una definición previa.
Para corregir el código, debes especificar el tipo de datos que se almacenarán en la lista. En este caso, dado que los valores que se están agregando a la lista son enteros, deberías usar `Integer` en lugar de `E`.

Pregunta 59

```java
public class Car {
    private void accelerate() {
        System.out.println("car accelerating");
    }

    private void brake() { // Nota: 'break' es una palabra reservada, debe ser 'brake'.
        System.out.println("car braking");
    }

    public void control(boolean faster) {
        if (faster)
            accelerate();
        else
            brake();
    }

    public static void main(String[] args) {
        Car car = new Car();
        car.control(false);
    }
}
// 'break' es una palabra reservada, debe ser 'brake'
```

La respuesta de la pregunta 59 es que el código no compilará debido a que la palabra clave `break` se usa incorrectamente como nombre de un método.

Pregunta 60

```java
class App {
    App() {
        System.out.println("1");
    }

    App(Integer num) {
        System.out.println("3");
    }

    App(Object num) {
        System.out.println("4");
    }

    App(int num1, int num2, int num3) {
        System.out.println("5");
    }

    public static void main(String[] args) {
        new App(100);
        new App(100L);
    }
}
// Salida: 3, 4
```

Respuesta: La salida del código es `3, 4`.

Explicación: Cuando se crea una nueva instancia de `App` con `new App(100)`, se llama al constructor que acepta un `Integer` (debido a la autoboxing de `int` a `Integer`), lo que imprime 3. Cuando se crea una nueva instancia con `new App(100L)`, se llama al constructor que acepta un `Object` (porque `100L` es un `Long`, que se puede tratar como un `Object`), lo que imprime 4.

Pregunta 61

```
class App {
    public static void main(String[] args) {
        int i = 42;
        String s = (i < 40) ? "life" : (i > 50) ? "universe" : "everything";
        System.out.println(s);
    }
}
// Salida: everything
```

Respuesta: La salida del código es `everything`.

Explicación: La expresión ternaria `(i < 40) ? "life" : (i > 50) ? "universe" : "everything"` evalúa `i`, que es 42. Como 42 no es menor que `40` y tampoco es mayor que `50`, la expresión resulta en `"everything"`, que se imprime.

Pregunta 62

```
class App {
    App() {
        System.out.println("1");
    }

    App(int num) {
        System.out.println("2");
    }

    App(Integer num) {
        System.out.println("3");
    }

    App(Object num) {
        System.out.println("4");
    }

    public static void main(String[] args) throws java.text.ParseException {
        String[] sa = {"333.6789", "234.111"};
        java.text.NumberFormat inf = java.text.NumberFormat.getInstance();
        inf.setMaximumFractionDigits(2);
        for (String s : sa) {
            System.out.println(inf.parse(s));
        }
    }
}
// java: unreported exception java.text.ParseException; must be caught or declared to be thr
```

Respuesta: El código no compilará debido a la excepción `java.text.ParseException`.

Explicación: El método `parse` de `NumberFormat` lanza una `ParseException`, que debe ser manejada con un bloque `try-catch` o declarada en la firma del método con `throws`. Como no se maneja ni se declara, el código falla en compilación.

Pregunta 63

```
class Y {
    public static void main(String[] args) {
        String s1 = "OCAJP";
        String s2 = "OCAJP" + "";
        System.out.println(s1 == s2);
    }
}
// Salida: true
```

Respuesta: La salida del código es `true`.

Explicación: s1 y s2 son literalmente iguales y el operador + no cambia el contenido, por lo que ambos apuntan al mismo objeto en el pool de cadenas. Por lo tanto, `s1 == s2` devuelve `true`.

Pregunta 64

```
class Y {
    public static void main(String[] args) {
        int score = 60;
        switch (score) {
            default:
                System.out.println("Not a valid score");
            case score < 70:
                System.out.println("Failed");
                break;
            case score >= 70:
                System.out.println("Passed");
                break;
        }
    }
}
```

Respuesta: El código no compilará.

Explicación: Los valores de `case` en un `switch` deben ser constantes, pero las expresiones como `score < 70` no son constantes. Esto resulta en un error de compilación porque no se pueden usar expresiones booleanas como etiquetas de `case`.

Pregunta 65

```
class Y {
    public static void main(String[] args) {
        int a = 100;
        System.out.println(-a++);
    }
}
// salida -100
```

Respuesta: La salida del código es `-100`.

Explicación: La expresión `-a++` usa el operador unario negativo y luego incrementa a después de la evaluación. Primero, `-a` es `-100`, y luego a se incrementa a `101`, pero el valor impreso es `-100`

Pregunta 66

```
class Y {
    public static void main(String[] args) {
        byte var = 100;
        switch (var) {
            case 100:
                System.out.println("var is 100");
                break;
            case 200:
                System.out.println("var is 200");
                break;
            default:
                System.out.println("In default");
        }
    }
}
```

Respuesta: El código no compilará debido a una conversión de tipos incompatible.

Explicación: `case 200` genera un error de compilación porque `200` está fuera del rango de valores que un `byte` puede representar (-128 a 127). Esto resulta en una posible pérdida de precisión en la conversión de `int` a `byte`

Pregunta 67

```
class Y {
    public static void main(String[] args) {
        A obj1 = new A();
        B obj2 = (B) obj1;
        obj2.print();
    }
}

class A {
    public void print() {
        System.out.println("A");
    }
}

class B extends A {
    public void print() {
        System.out.println("B");
    }
}
```

Respuesta: Se lanza una `ClassCastException`.

Explicación: Intentar hacer un cast de `A` a `B` (`B obj2 = (B) obj1;`) fallará en tiempo de ejecución porque `obj1` es una instancia de `A`, no de `B`. Esto causa una `ClassCastException`.

Pregunta 68

```
class Y {
    public static void main(String[] args) {
        String fruit = "mango";
        switch (fruit) {
            default:
                System.out.println("ANY FRUIT WILL DO");
            case "Apple":
                System.out.println("APPLE");
            case "Mango":
                System.out.println("MANGO");
            case "Banana":
                System.out.println("BANANA");
                break;
        }
    }
}
// Salida: ANY FRUIT WILL DO APPLE MANGO BANANA
```

Respuesta: La salida del código es `ANY FRUIT WILL DO APPLE MANGO BANANA`.

Explicación: El `switch` comienza con el `default` y no hay `break` hasta el final del `case "Banana"`. Esto provoca la ejecución de todos los casos consecutivos (`fall-through`).

Pregunta 69

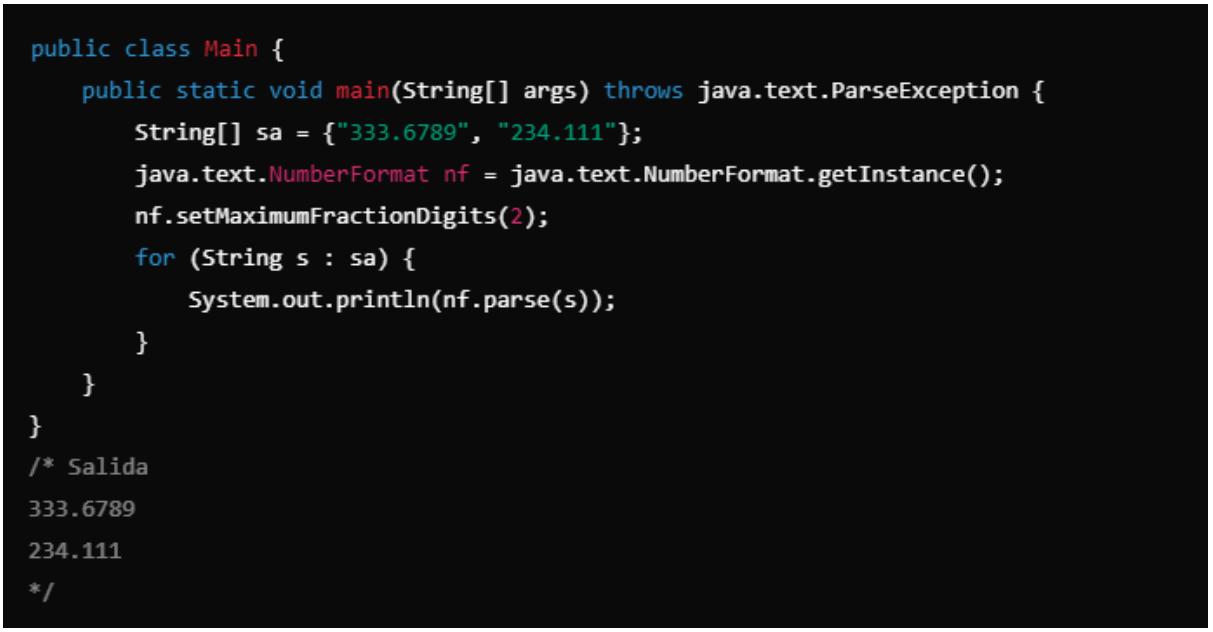
```
abstract class Animal {
    private String name;

    Animal(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
}

class Dog extends Animal {
    private String breed;

    Dog(String breed) {
        super(null); // Añadido para corregir el error
        this.breed = breed;
    }

    Dog(String name, String breed) {
        super(name);
        this.breed = breed;
```

```
    }

    public String getBreed() {
        return breed;
    }
}

class Test {
    public static void main(String[] args) {
        Dog dog1 = new Dog("Beagle");
        Dog dog2 = new Dog("Bubbly", "Poodle");
        System.out.println(dog1.getName() + ":" + dog1.getBreed() + ":" + dog2.getName() + ":"
+ dog2.getBreed());
    }
}
// compilation fails
```

Respuesta: El código no compilará.

Explicación: El constructor `Dog(String breed)` no llama al constructor de la clase abstracta `Animal`. Debe usar `super` para inicializar la parte de `Animal` de `Dog`.

Pregunta 70



```
public class Main {
    public static void main(String[] args) throws java.text.ParseException {
        String[] sa = {"333.6789", "234.111"};
        java.text.NumberFormat nf = java.text.NumberFormat.getInstance();
        nf.setMaximumFractionDigits(2);
        for (String s : sa) {
            System.out.println(nf.parse(s));
        }
    }
}
/* Salida
333.6789
234.111
*/
```

Respuesta: La salida es `333.6789` `234.111`.

Explicación: Aunque se ha configurado `setMaximumFractionDigits(2)`, `NumberFormat.parse` no redondea o trunca el número, sino que lo interpreta como un `Number`. La salida muestra los números originales.

Pregunta 71

```java
public class Main {
    public static void main(String[] args) {
        java.util.Queue<String> products = new java.util.ArrayDeque<String>();
        products.add("p1");
        products.add("p2");
        products.add("p3");
        System.out.println(products.peek());
        System.out.println(products.poll());
        System.out.println("");
        products.forEach(s -> {
            System.out.println(s);
        });
    }
}
/* Salida
p1
p1

p2
p3
*/
```

Respuesta: La salida es p1 p1 p2 p3.

Explicación: `peek` obtiene pero no elimina el primer elemento (p1). `poll` obtiene y elimina el primer elemento (p1). Luego, el método `forEach` imprime los elementos restantes (p2 y p3).

Pregunta 72

```java
public class Main {
    public static void main(String[] args) {
        System.out.println(2 + 3 + 5);
        System.out.println("+" + 2 + 3 * 5);
    }
}
// Salida: 10 +215
```

Respuesta: La salida del código es `10 +215`.

Explicación: `2 + 3 + 5` se evalúa aritméticamente, resultando en `10`. En `"+" + 2 + 3 * 5`, la multiplicación se evalúa primero (`3 * 5` es 15), y luego se concatena a la cadena, resultando en +215.