1. ¿Que arroja?

```
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;
                  }
                }
            }
    }
}
```

1 FINN 2 FINN 2 JAKE 3 FINN 3 JAKE
Cuando x es 1:

Imprime "1 FINN".

Luego, el break hace que salga del bucle interno, por lo que no se imprime "1 JAKE".

Cuando x es 2:

Imprime "2 FINN".

Luego, imprime "2 JAKE".

Cuando x es 3:

Imprime "3 FINN".

Luego, imprime "3 JAKE"

2. ¿Que 5 lineas son correctas?

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

`private int lightsaber (int x){return 0;}` //Error el modificador de acceso en la clase derivada no puede ser más restrictivo que el modificador de acceso en la clase base

`protected int lightsaber (long x){return 0;}` // Correcto. Sobreescritura de método adecuada, por cambio de parámetro

`private int lightsaber (long x){return 0;}` // Correcto No se esta sobreescribiendo el método, al tener otro parámetro se trata de un método independiente

protected long lightsaber (int x){return 0;} // Error Para que la sobrescritura sea válida, los métodos deben tener la misma firma, incluyendo el tipo de retorno.

protected long lightsaber (int x, int y){return 0;} //Correcto

public int lightsaber (int x){return 0;} // Correcto

protected long lightsaber (long x){return 0;} // Valido por ser sobrecarga de metodo

3. ¿Que resultado arroja?

```
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

Crea una instancia de Mouse usando new Mouse(15), lo que usa el primer constructor. Este constructor invoca el segundo constructor, que a su vez invoca el tercer constructor.

Después de la inicialización, el objeto mouse tendrá weight de 15, numTeeth de 16 y numWhiskers de 6.

Llama al método print() para mostrar estos valores.

4. ¿Cual es la salida?

```
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();
    }
}
```

// arachnid spider s s

Cuando el constructor Spider se ejecuta, primero se llama al constructor Arachnid, que imprime "arachnid".

Luego se ejecuta el constructor Spider, que imprime "spider".

Después de la construcción del objeto Spider, el método run() se llama:

Dentro del método run(), this.type se establece en "s".

super.type se refiere a la misma variable type en la instancia actual de Spider, ya que type es una variable de instancia, no una variable estática o local.

Por lo tanto, tanto this.type como super.type muestran "s".

5. Resultado

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

// Respuesta correcta: 997, 997

Se inicializa la variable ov con el valor 999.

ov--; //Se decrementa ov en 1, así que ov pasa de 999 a 998.

System.out.println(--ov); // --ov es un decremento prefijo, lo que significa que ov se decrementa antes de que se use en la impresión. Así que ov pasa de 998 a 997, y luego se imprime 997.

System.out.println(ov); // este punto, ov ya se ha decrementado a 997, así que se imprime 997.

6. Resultado

```
class Overloading {

    public static void main(String[] args) {
      System.out.println(overload("a"));
      System.out.println(overload("a", "b"));
      System.out.println(overload("a", "b", "c"));
    }

    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

overload("a") llama al método overload(String s) y devuelve "1".

overload("a", "b") llama al método overload(String s, String t) y devuelve "4".

overload("a", "b", "c") llama al método overload(String... s) que es el método varargs. Este método puede aceptar cualquier número de argumentos de tipo String, incluyendo tres. y devuelve "2".

7. Resultado

```
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(); // Intento de casting
    }
}
```

// ClassCastException:

En Java, el casting de un objeto a una clase que no es parte de su jerarquía de herencia causará una excepción en tiempo de ejecución (ClassCastException).

En este caso, obj es una instancia de Base1, por lo que no puede ser convertido a Base2, ya que Base1 y Base2 son dos clases distintas sin relación directa en la jerarquía de herencia (excepto que ambas extienden Base, son hermanos).

8. Resultado

```
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);
    }
}
```

// El código no compila

No puedes asignar un valor numérico a una variable de tipo cadena. Esto causará un error de compilación.

9. Resultado

```
public class NumberPrint {
    public static void main(String[] args) {
        int number1 = 0b0111;
        int number2 = 0111_000;

        System.out.println("Number1: " + number1);
        System.out.println("Number2: " + number2);
    }
}
```

//Salida: 7 7 ojo que imprime dos veces number 1

La notación 0b (o 0B) se usa para representar números en binario.

0111 en binario se convierte a decimal de la siguiente manera:

0b01110b0111 = 0×23+1×22+1×21+1×200×23+1×22+1×21+1×20

= 0+4+2+10+4+2+1

= 77

Por lo tanto, number1 tiene el valor 7.

10. Resultado

```
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

La variable de instancia num se inicializa con 100.

Después de llamar a calc(2), num se actualiza a 20 (2 * 10). this.num se refiere a la variable de instancia de la clase, mientras que num (sin this) se refiere al parámetro local del método.

El método printNum() imprime este valor actualizado, por lo tanto, la salida es 20.

11. Que Aseveraciones son correctas

```
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":

Correcto. Si omites la declaración import java.util.Random;, el código no compilará, ya que la clase Random pertenece al paquete java.util. Si no se importa explícitamente, el compilador no sabrá a qué clase Random te refieres y generará un error de compilación.

"java.lang and util.random are redundant":

Incorrecto. La declaración es incorrecta por varias razones:

java.lang es un paquete que se importa automáticamente, y no necesitas declararlo manualmente.

util.random no es un paquete válido. La clase Random pertenece a java.util.Random, y no hay un paquete llamado util.random. La aseveración también es incorrecta porque java.lang y java.util.Random no están relacionados ni son redundantes.

"You don't need to import java.lang":

Correcto. El paquete java.lang se importa automáticamente en todas las clases de Java. Este paquete incluye clases fundamentales como String, System, Math, etc., por lo que no necesitas importar explícitamente este paquete.

12. Resultado

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

//salida: 10, 12

La primera llamada a System.out.println(var++) imprime 10, luego var se incrementa a 11.

La segunda llamada a System.out.println(++var) incrementa var a 12 antes de imprimirlo, por lo que imprime 12

13. Resultado

```
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; Respuesta correcta mn = 11

El bucle for itera desde hr = 11 hasta hr = 7, incrementando sg en cada iteración. Cuando hr llega a 6, el bucle se detiene y el valor final de sg es 5, que se imprime como salida.

14. Cuáles son verdad

- "An ArrayList is mutable":

Verdadero. Un ArrayList es mutable, lo que significa que puedes cambiar su contenido después de haber sido creado (puedes agregar, eliminar o modificar elementos).

- "An Array has a fixed size":

Verdadero. Un array en Java tiene un tamaño fijo, lo que significa que una vez creado, no puedes cambiar su tamaño.

- "An array is mutable":

Verdadero. Un array en Java es mutable en el sentido de que puedes cambiar los valores de sus elementos después de haber sido creado.

- "An array allows multiple dimensions":

Verdadero. Un array en Java puede tener múltiples dimensiones, lo que significa que puedes crear arrays de arrays, como en int[][] para una matriz bidimensional.

- "An ArrayList is ordered":

Verdadero. Un ArrayList es ordenado, lo que significa que los elementos se mantienen en el orden en que fueron añadidos.

- "An array is ordered":

Verdadero. Un array en Java también es ordenado, lo que significa que los elementos se almacenan en el orden en que se insertan y se accede a ellos mediante su índice.

15. Resultado

```
public class MultiverseLoop {
```

```
    public static void main(String[] args) {
        int negotiate = 9;
        do {
            System.out.println(negotiate);
        } while (--negotiate);
    }
}
```

La condición del bucle while espera un valor booleano, pero --negotiate es una operación que devuelve un valor entero, no un booleano.

En Java, un valor entero no puede ser implícitamente convertido a un booleano. Como resultado, este código no compilará.

16. Resultado

```
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));
    }
}
```

Exception at runtime. El método filter se utiliza para filtrar los elementos del stream, pero en este caso, los elementos filtrados no se almacenan ni se encadenan, lo que significa que la operación filter no tiene efecto. Además, el stream se consume después de aplicar filter, por lo que no puedes utilizar el mismo stream para llamar a forEach.

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?

x es de tipo Object: Dado que todas las clases implícitamente extienden Object y todas las interfaces también implícitamente extienden Object, puedes asignar cualquier objeto a una variable de tipo Object. En este caso, y, siendo una interfaz, es compatible con Object, por lo que la asignación x = y; es legal.

x es una variable de un tipo que es una superclase o una clase que implementa y: Si y es una instancia de una clase que implementa la interfaz SomeInterface, y x es una variable de un tipo que es una superclase o una clase que y implementa, entonces la asignación es legal.

18. Pregunta

When a byte is added to a char, what is the type of the result?

Cuando se suma un byte a un char en Java, el tipo del resultado es int.

En Java, cuando se realizan operaciones aritméticas entre tipos primitivos, Java primero promueve los operandos a un tipo común antes de realizar la operación. Para operaciones aritméticas, los tipos byte, short y char se promueven a int.

19. Pregunta

The standard application programming interface for accessing databases in Java?

JDBC

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);

23. Pregunta

```
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) {
            throw new RuntimeException();
        }
    }
}
```

En el método doSomething(), se declara que puede lanzar una excepción IOException con la firma throws IOException. Sin embargo, dentro de este método, solo se lanza una excepción de tipo RuntimeException. La declaración throws IOException no está relacionada con el código actual y es innecesaria en este contexto. La excepción IOException no se lanza en el método, y RuntimeException es una excepción no comprobada, lo que significa que no es necesario declarar su lanzamiento en la firma del método

24. Resultado

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

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

En Java, cuando una clase implementa una interfaz, todos los métodos de la interfaz deben ser implementados en la clase concreta. Los métodos de una interfaz tienen una visibilidad pública (public), y la implementación en la clase concreta también debe ser pública.

Interfaz Interviewer: Declara un método interviewConducted() que es implícitamente public y abstract.

Clase Manager: Implementa el método interviewConducted(), pero no lo declara como public.

25. Pregunta

```
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);
        spider.printName(9.0);
    }
}
```

Salida: Spider, Arth

spider.printName(4): El argumento 4 es un int. Dado que la clase Spider tiene un método printName(int input), este método se llama y se imprime "Spider".

spider.printName(9.0): El argumento 9.0 es un double. El método printName(double input) heredado de Arthropod se llama, y se imprime "Arth".

26. Pregunta

```
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]);
        }
    }
}
```
Salida: Wed Wed Wed

Days.values(): Este método devuelve un arreglo con todos los valores de la enumeración en el orden en que están definidos.

En el bucle for-each, d toma cada uno de los valores de la enumeración (Mon, Tue, Wed), pero no se utiliza en el cuerpo del bucle.

Days[] d2 = Days.values();: Obtiene un arreglo con los valores de la enumeración.

System.out.println(d2[2]);: Imprime el valor en la posición 2 del arreglo d2. Dado que los valores de la enumeración están en el índice 0, 1 y 2, d2[2] corresponde a Wed.

El bucle for-each itera tres veces, pero en cada iteración imprime el valor en d2[2], que siempre es Wed.

27. Pregunta

```
public class Main {
    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);
    }
}
```
// Salida: true 20 false

y != 10: Esto se evalúa a true porque 20 no es igual a 10.

z = false: Esto asigna false a z. La operación de asignación devuelve el valor asignado, que en este caso es false.

true ^ false: El operador ^ es el operador XOR (o exclusivo). El resultado de true ^ false es true porque XOR devuelve true si uno de los operandos es true y el otro es false.

28. Pregunta

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

    static void add(int num) {
```

```
        System.out.println(num + "");
    }

    InitializacionOrder() {
        add(5);
    }

    static {
        add(4);
    }

    {
        add(6);
    }

    static {
        new InitializacionOrder();
    }

    {
        add(8);
    }

    public static void main(String[] args) {
    }
}
// Salida: 2 4 6 8 5
```

Bloques Estáticos:

Los bloques estáticos se ejecutan en el orden en que aparecen en la clase, una sola vez cuando se carga la clase por primera vez.

Primer Bloque Estático: add(2); imprime 2.

Segundo Bloque Estático: add(4); imprime 4.

Tercer Bloque Estático: new InitializacionOrder(); crea una instancia de InitializacionOrder, lo que desencadena la ejecución de bloques de instancia y el constructor.

Bloques de Instancia y Constructor:

Los bloques de instancia se ejecutan cada vez que se crea una nueva instancia de la clase, en el orden en que aparecen, antes del constructor.

Primer Bloque de Instancia: add(6); imprime 6.

Constructor: add(5); imprime 5.

Segundo Bloque de Instancia: add(8); imprime 8.

29. Pregunta

```
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");
        }
    }
}
```

Salida: They don't match

String message1 = "Wham bam";: Esto crea una cadena literal en el pool de cadenas de Java.

String message2 = new String("Wham bam");: Esto crea una nueva instancia de String en el heap, aunque el contenido de la cadena es el mismo que message1.

message1 != message2: Este operador compara las referencias de los objetos, no sus valores. Dado que message1 apunta a una cadena en el pool de cadenas y message2 es una nueva instancia en el heap, message1 y message2 no son la misma referencia. Por lo tanto, message1 != message2 es true.

30. Pregunta

```
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

System.out.println("1");: Imprime 1.

Entra en el bloque try:

System.out.println("2");: Imprime 2.

name.toString();: Intenta llamar a toString() en name, que es null. Esto lanza una NullPointerException.

System.out.println("3");: Este código no se ejecutará porque la excepción se lanza antes de llegar a esta línea.

El bloque catch captura la NullPointerException:

System.out.println("4");: Imprime 4.

throw e;: Relanza la NullPointerException, lo que provoca que el método run() termine con una excepción no manejada.

Después de run():

El método run() lanza una excepción que no es capturada en main, por lo que el programa se detiene.

System.out.println("6");: Este código no se ejecutará porque el programa se interrumpe debido a la excepción no manejada

31. Pregunta

```
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

Intenta usar JDBC (Java Database Connectivity) para realizar una operación en una base de datos.

32. Pregunta

```
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() + "," + ab3.getHash());
    }

    public int getHash() {
        return 676000;
    }
}

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

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

// Salida: the profits are 676000, 18330000, 27980000

ab1.getHash() llama al método getHash() de MarvelClass, que retorna 676000.

ab2.getHash() llama al método getHash() de MarvelMovieA, que retorna 18330000.

ab3.getHash() llama al método getHash() de MarvelMovieB, que retorna 27980000.

Aunque ab1, ab2, y ab3 son de tipo MarvelClass, el método getHash() que se invoca es el de la clase concreta a la que el objeto pertenece, gracias al polimorfismo.

33. Pregunta

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

// Salida: An ArrayIndexOutOfBoundsException

El bucle for está definido con i <= arr.length. Esto significa que i iterará desde 0 hasta arr.length, que es 4 en este caso (ya que el array tiene 4 elementos).

Los índices válidos para acceder a los elementos del array arr son 0 a 3, ya que el índice máximo es arr.length - 1.

Cuando i es igual a 4, se intentará acceder a arr[4]. Sin embargo, el índice 4 está fuera del rango válido del array arr, ya que el último índice válido es 3.

Acceder a un índice fuera del rango del array (arr[4] en este caso) provocará una excepción ArrayIndexOutOfBoundsException.

// 34. Pregunta

```
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++;
            }
        }
    }
}
/*
```

Salidas:

beans,2

beans,3

egg,2

egg,3

ham,2

ham,3

juice,2

juice,3

*/

El bucle for-each itera sobre cada elemento del array breakfast.

Dentro del bucle for-each, el bucle while imprime el elemento actual del array junto con los números 2 y 3, que corresponden a los valores de dish cuando dish es menor que el tamaño del array.

35. Pregunta

Which of the following statements are true:

• StringBuilder es generalmente más rápido que StringBuffer.

StringBuilder es generalmente más rápido que StringBuffer porque StringBuilder no está sincronizado. La sincronización es una característica que hace que StringBuffer sea seguro para el uso en múltiples hilos al asegurar que solo un hilo pueda acceder al objeto a la vez. Sin embargo, esta sincronización adicional introduce una sobrecarga que puede afectar el rendimiento.

• StringBuffer is threadsafe; StringBuilder is not.

StringBuffer es thread-safe porque sus métodos están sincronizados, lo que significa que puede ser utilizado de forma segura en entornos multi-hilo. En contraste, StringBuilder no tiene sincronización, lo que lo hace no thread-safe pero más eficiente en situaciones donde no se necesita la seguridad en entornos multi-hilo.

36. Pregunta

```
class CustomKeys {
    Integer key;

    CustomKeys(Integer k) {
        key = k;
    }

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

Salida: Compilation fail

37. Pregunta

The catch clause is of the type:

• **Throwable**. Captura todas las excepciones y errores.

• **Exception but NOT including RuntimeException**. Captura excepciones comprobadas, excluyendo excepciones no comprobadas.

• **CheckedException**. Captura excepciones comprobadas (concepto similar a la opción anterior, aunque CheckedException no es un tipo específico de excepción en Java).

• **RuntimeException**. Captura excepciones no comprobadas.

• **Error**. Captura errores graves (aunque generalmente no se recomienda).

38. Pregunta

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.

El bucle for-each en Java, también conocido como "enhanced for loop", es una forma simplificada de iterar a través de colecciones de elementos como arrays, listas, conjuntos y otros tipos de colecciones. Introducido en Java 5, este tipo de bucle facilita la iteración al eliminar la necesidad de usar un índice o un iterador explícito.

No se puede usar para modificar la colección: No puedes usar el bucle for-each para eliminar o agregar elementos a la colección durante la iteración, ya que esto puede causar una ConcurrentModificationException.

No se puede obtener el índice: No puedes obtener el índice del elemento actual directamente; solo tienes acceso al valor del elemento.

39. Pregunta

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

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

40. Pregunta

```
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");
        }
    }
}
// Salida: Equal
```

s1.equalsIgnoreCase(s2): El método equalsIgnoreCase() compara las dos cadenas s1 y s2 sin considerar las diferencias de mayúsculas y minúsculas. Dado que "Java" y "java" son iguales si ignoramos las diferencias de caso, el resultado será true.

41. Pregunta

```
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);
        }
    }
}
/*
```

Salidas:

apple

banana

grapes

pears

*/

La expresión lambda (a, b) -> a.compareTo(b) compara dos cadenas de acuerdo con el orden lexicográfico, que es el orden alfabético para cadenas de texto.

El método sort modifica el array fruits en su lugar, por lo que el array fruits estará ordenado después de llamar a sort().

42. Pregunta

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

// Salida: This code will throw an ArrayIndexOutOfBoundsException

En Java, intentar acceder a un índice fuera del rango de un array (ya sea negativo o mayor que el tamaño del array) resulta en una excepción ArrayIndexOutOfBoundsException. En este caso, el índice -1 está fuera del rango válido.

43. Pregunta

```
class Salmon {
    int count;

    public void Salmon() {
        count = 4;
    }

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

Salida:0

Dado que el método Salmon() no es un constructor válido, no se llama automáticamente cuando se crea una nueva instancia de Salmon. En su lugar, se ejecuta el constructor predeterminado proporcionado por Java, que no inicializa count.

44 pregunta

```
class Circuit {
    public static void main(String[] args) {
        runlap();
        int c1 = c2;
        int c2 = v;
    }

    static void runlap() {
        System.out.println(v);
    }

    static int v;
}
```

// corregir linea 6; c1 se le asigna c2 pero c2 aun no se declara

// 45 pregunta

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

// salida: 611 (11+20*30)

Preincremento de a (++a): a se incrementa en 1 antes de ser utilizado en la expresión. Por lo tanto, a pasa de 10 a 11.

Postincremento de b (b++): b se utiliza en la expresión antes de ser incrementado, por lo que en la multiplicación se usará el valor original de b que es 20. Después de la multiplicación, b se incrementará en 1, pero este nuevo valor no afecta el cálculo actual.

Multiplicación (b * c): El valor de b (20) se multiplica por el valor de c (30), lo que da 600.

Suma (++a + (b * c)): Finalmente, el valor de a (11) se suma al resultado de la multiplicación (600), lo que da 611.

// 46 pregunta

```
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
En Java, cuando se usa un varargs (como String... y), éste debe ser el último parámetro en la lista de parámetros del método. Esto se debe a que un varargs puede aceptar un número variable de argumentos, y Java necesita saber cuáles van a ser esos argumentos al final. En este caso, el parámetro int x está después del varargs, lo cual no es permitido y causará un error de compilación.

// 47 pregunta
```
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");
        new Tree("leaves");
    }

    public static void main(String[] args) {
        new Forest();
    }
}
```
/*plant
leaves
plant
leaves*/
Llamada a super("leaves") en el constructor de Forest:

Constructor de Plant se ejecuta → Imprime "plant"

Constructor de Tree se ejecuta → Imprime "leaves"

Creación de la nueva instancia de Tree dentro del constructor de Forest:

Constructor de Plant se ejecuta → Imprime "plant"

Constructor de Tree se ejecuta → Imprime "leaves"

// 48 pregunta

```
class Test {
    public static void main(String[] args) {
        String s1 = "hello";
        String s2 = new String("hello");
        s2 = s2.intern(); // el intern() asigna el mismo hash
conforme a la cadena
        System.out.println(s1 == s2);
    }
}
```
// Salida: true

El método intern() busca la cadena "hello" en el String Pool. Si la cadena ya existe en el String Pool, intern() devuelve la referencia a esa cadena en el pool. Como "hello" ya existe en el pool (por s1), s2 ahora apuntará a la misma instancia de cadena que s1.

Aquí se está comparando si s1 y s2 apuntan al mismo objeto en memoria. Dado que después de la llamada a intern(), s2 apunta al mismo objeto en el String Pool que s1, la comparación s1 == s2 es true.

// 49 pregunta
Cuál de las siguientes construcciones es un ciclo infinito while:
// • while(true);
// • while(1==1){}

// 50 pregunta
```
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);
    }
}
```
// Salida: 0 -> cero
Se evalúa si a == b. Como 10 no es igual a 37, este bloque se omite.
Luego, se evalúa si a > b. Como 10 no es mayor que 37, este bloque también se omite.
Como ninguna de las condiciones se cumple, z permanece con su valor inicial, que es 0.

// 51 pregunta

```
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();
        c.name = "Oracle";
    }
}
```
// Exception StackOverflowError
En este constructor, se crea un nuevo objeto course llamado c, y se asigna el valor "Oracle" a su campo name. Sin embargo, este objeto c es una variable local dentro del constructor y no tiene ningún efecto sobre el objeto course original que se está creando en Main. Este código causa una recursión infinita al intentar crear constantemente nuevos objetos course, lo que eventualmente llevará a un desbordamiento de pila (StackOverflowError).

// 52 pregunta
```
public class Main {
    public static void main(String[] args) {
        String a;
        System.out.println(a.toString());
  }
}
```
// builder fails

Aquí, a es declarada como una variable de tipo String, pero no se le asigna ningún valor. En Java, las variables locales deben ser inicializadas antes de ser usadas. Si intentas usar una variable local sin inicializarla, el compilador lanzará un error.

En esta línea, estás tratando de invocar el método toString() en la variable a, pero como a no ha sido inicializada, el código no compilará. Java no permite el uso de variables locales no inicializadas.

// 53 pregunta
```
public class Main {
    public static void main(String[] args) {
        System.out.println(2 + 3 + 5);
        System.out.println("+" + 2 + 3 + 5);
```

```
    }
}
```
// salida 10 + 235
En este caso, la primera parte de la expresión es una cadena (String), lo que desencadena la concatenación de cadenas en lugar de la suma aritmética. Java concatenará los valores en el orden en que aparecen:

"+" + 2 → "+2"
"+2" + 3 → "+23"
"+23" + 5 → "+235"

Por lo tanto, esta línea imprimirá "+235"

// 54 pregunta
```
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");
    }
}
```
// salida: Here1 , here 3

Aquí, se verifica si a es igual a b. Como ambos tienen el valor 2, esta condición es verdadera, por lo que se imprimirá "Here1".

Aquí, se verifica si a es diferente de b. Dado que a es igual a b (ambos son 2), esta condición es falsa, por lo que no se imprimirá "here2".

Aquí, se verifica si a es mayor o igual a b. Como a es igual a b (ambos son 2), esta condición es verdadera, por lo que se imprimirá "Here3".

// 55 pregunta
```
public class Main extends count {
    public static void main(String[] args) {
        int a = 7;
        System.out.println(count(a, 6));
    }
}

class count {
```

```
    int count(int x, int y) {
        return x + y;
    }
}
```
// builder fails

En este contexto, count se está llamando como si fuera un método estático, pero en realidad es un método de instancia de la clase count. Los métodos de instancia deben ser llamados a través de una instancia de la clase. Además, el método count no es estático, por lo que no puede ser llamado directamente desde un contexto estático (como main) sin una instancia de la clase count.

// 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");
    }
}
```
// Salida: magic town

Cuando se ejecuta el programa, Java buscará el método public static void main(String[] args) como punto de entrada y ejecutará ese método.

// 57 pregunta

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

Primera línea System.out.println(a++ + 2);:

Postincremento (a++): El operador ++ en a++ incrementa el valor de a después de que se ha utilizado en la expresión. Por lo tanto, el valor de a en esta expresión es 0 y luego a se incrementa a 1.

Cálculo: La expresión a++ + 2 se evalúa como 0 + 2, lo que resulta en 2.

Resultado: La salida de esta línea es 2. Después de esta línea, el valor de a se incrementa a 1.

Segunda línea System.out.println(a);:

En esta línea, simplemente se imprime el valor actual de a.

Después de la primera línea, a ha sido incrementado a 1.

Por lo tanto, la salida de esta línea es 1.

// 58 pregunta

```
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);
    }
}
```

// builder fails

List<E> p = new ArrayList<>();

Aquí, E es un tipo genérico que no ha sido definido en el código. Los tipos genéricos como E se usan en contextos donde se especifican tipos, como en clases o métodos genéricos, pero no se pueden usar directamente sin una declaración concreta.

El código intenta agregar valores enteros (int) a una lista que tiene un tipo genérico no definido. Esto no es válido ya que E no se ha definido y la lista p no sabe qué tipo de elementos debe contener.

// 59 pregunta

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

    private void break() {
        System.out.println("car breaking");
    }

    public void control(boolean faster) {
        if (faster == true)
            accelerate();
        else
            break();
    }

    public static void main(String[] args) {
        Car car = new Car();
```

```
        car.control(false);
    }
}
```
// break es una palabra reservada
Dado que car.control(false) es llamado, faster es false, así que el código dentro del bloque else se ejecutará.
Esto llamará al método break(), que imprime "car breaking".

// 60 pregunta
```
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 …

El valor 100 es un literal int.

El constructor más específico que coincide con un int es App(int num1, int num2, int num3), pero este requiere tres argumentos, no uno.

El siguiente constructor que acepta un int como argumento es App(Integer num), que es un constructor que acepta un Integer.

En Java, el compilador puede hacer un boxing automático de int a Integer, así que el constructor App(Integer num) será elegido aquí.

Por lo tanto, se imprimirá "3".

Instancia new App(100L);:

El valor 100L es un literal long.

No hay un constructor que acepte directamente un long. El compilador debe hacer una conversión.

El constructor más adecuado para un long es App(Object num), ya que long se puede convertir a Object sin problemas.

Por lo tanto, se imprimirá "4".

// 61 pregunta

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

// Salida: everething

i tiene el valor 42.

La primera condición i < 40 es false (porque 42 no es menor que 40).

Por lo tanto, se evalúa la segunda parte del operador ternario: (i > 50) ? "universe" : "everething".

La condición i > 50 también es false (porque 42 no es mayor que 50).

Dado que esta condición es false, se selecciona "everething".

// 62 pregunta

```
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) {
        String[] sa = {"333.6789", "234.111"};
        NumberFormat inf = 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 thrown

// 63 pregunta
```
class Y {
    public static void main(String[] args) {
        String s1 = "OCAJP";
        String s2 = "OCAJP" + "";
        System.out.println(s1 == s2);
    }
}
```
// salida: true

// 64 pregunta
```
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;
        }
    }
}
```
// salida: Error de compilacion - java: reached end of file while parsing

// 65 pregunta
```
class Y {
    public static void main(String[] args) {
        int a = 100;
        System.out.println(-a++);
    }
}
```
// salida -100

La expresión -a++ se evalúa primero con el valor actual de a, que es 100, y luego se aplica la negación, resultando en -100.

Después de esta operación, el valor de a se incrementará a 101, pero eso no afecta el valor que se imprime en esta línea.

// 66 pregunta
```
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");
        }
    }
}
```
// salida: Error de compilacion - java: incompatible types: possible lossy conversion from int to byte

// 67 pregunta
```
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");
    }
}
```
// ClassCastException
Después del casting incorrecto, obj2 se refiere a obj1, que sigue siendo un objeto de tipo A, no de tipo B.
Si el casting fuera válido, se llamaría al método print() de B. Sin embargo, debido al casting incorrecto, esto causará una ClassCastException en tiempo de ejecución antes de que se pueda ejecutar la llamada a print()

// 68 pregunta
```
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;
        }
    }
}
```

Evaluación del switch:

El switch no encuentra un case que coincida exactamente con "mango", por lo que se ejecuta el bloque default:.

Después del bloque default:, se ejecutan todos los bloques de case que siguen, debido al fallthrough:

"ANY FRUIT WILL DO" (del default case)

"APPLE" (del case "Apple")

"MANGO" (del case "Mango")

"BANANA" (del case "Banana")

El break al final de case "Banana": detiene la ejecución del swi

// 69 pregunta

```
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) {
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
El constructor Dog(String breed) no llama al constructor de la superclase Animal. Esto resultará en un error de compilación porque la clase Animal no tiene un constructor sin parámetros.

//70 Pregunta
```
public class Main {
public static void main(String[] args) throws ParseException {
String[]sa = {"333.6789","234.111"};
NumberFormat nf = NumberFormat.getInstance();
nf.setMaximumFractionDigits(2);
for (String s: sa
) {

System.out.println(nf.parse(s));
}
}
}
```
/*Salida
333.6789
234.111
*/
Configuración de NumberFormat:

NumberFormat nf = NumberFormat.getInstance(); obtiene una instancia de NumberFormat para el formato de números.

nf.setMaximumFractionDigits(2); establece que el número máximo de dígitos después del punto decimal será 2.

Análisis de cadenas:

nf.parse(s) convierte la cadena s en un objeto Number, aplicando la configuración de dígitos fraccionarios.

Ejemplo de análisis:

Para la cadena "333.6789", nf.parse(s) convertirá el número a 333.68 debido al redondeo a 2 dígitos decimales.

Para la cadena "234.111", nf.parse(s) convertirá el número a 234.11 debido al redondeo a 2 dígitos decimales.