

# BANCO DE PREGUNTAS JAVA

## 1. ¿QUÉ 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;  
                }  
            }  
        }  
    }  
}
```

### EXPLICACIÓN:

Tenemos un arreglo de String de tamaño 2, tenemos un ciclo for dentro de él hay un foreach que se ejecutará por cada iteración del ciclo for el cual tiene una condición que dice que si x es igual a 1 se rompe el ciclo foreach y continua el ciclo for, por lo que no imprimirá 1JAKE.

## 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: sobrescritura de método adecuada, por cambio de parámetro.  
    private int lightsaber(long x) { return 0; } // Correcto: no se está sobrescribiendo 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; } // Correcto por ser sobrecarga de método.  
}
```

### 3. ¿QUÉ 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

#### EXPLICACIÓN:

La clase Mouse tiene tres constructores adaptados para recibir al menos un parámetro en este caso el peso, este constructor invoca otro constructor que recibe dos parámetros y se asigna un valor al número de dientes, y a su vez se invoca el constructor con tres parámetros e igual se manda un valor definido para el último atributo, asigna el valor a cada atributo por lo que puede imprimir los tres valores con solo haber inicializado la instancia con 15.

#### 4. ¿CUÁL 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

#### EXPLICACIÓN:

Se crea un objeto de la clase Spider que hereda de arachnid, por lo que el constructor de la clase Spider invoca el constructor de Arachnid y en el método run al ser type una variable de instancia se refieren a lo mismo.

## 5. ¿CUÁL ES EL 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);  
    }  
}
```

// Respuesta correcta: 997, 997

### EXPLICACIÓN:

Son dos clases que al momento de compilar se crearán dos archivos .class por lo que se ejecutan independientemente, ambos códigos decrementan el valor de la variable en 2 unidades, en la primera impresión se usa un pre-increment por lo que resta una unidad antes de ejecutar la impresión, por lo que para la clase Test, se imprimirá: 2 2; y para la clase Sheep se imprimirá: 997 997.

## 6. ¿CUÁL ES EL 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

### EXPLICACIÓN:

El código muestra como es la sobrecarga de métodos, respetando el nombre del método mientras cambia la lista de parámetros, primero se ejecuta el que recibe un String, imprimiendo "1", luego el método que recibe 2 Strings e imprime 4, el ultimo método invocado hace referencia al método que hace uso de los varargs que recibe una lista de parámetros del mismo tipo recibéndolo como un arreglo, por lo que retorna 2 y es el valor impreso.

## 7. RESULTADO

```
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();  
    }  
}
```

// ClassCastException

### EXPLICACIÓN:

Se produce un ClassCastException, porque en la jerarquía de herencia están en el mismo nivel Base1 y Base2, por lo que por el principio de herencia cada clase tendría su propia especificidad, tendría que ser el caso que Base1 extiende de Base2 para poder hacer ese cast.

## 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

### EXPLICACIÓN:

En el código se declaran 3 variables locales una de tipo int y dos de tipo String, en el String anotherFish se intenta asignar el resultado de la suma de dos enteros, operación que no es válida ya que intentara convertir el int en String y no podrá compilar, una forma de poder realizarlo sería con un Integer: new Integer(numFish + 1).toString();

## 9. ¿CUÁL ES EL RESULTADO?

```
class MathFun {  
    public static void main(String[] args) {  
        int number1 = 0b0111;  
        int number2 = 0111_000;  
  
        System.out.println("Number1: " + number1);  
        System.out.println("Number2: " + number1);  
    }  
}
```

Salida:

Number1: 7

Number2: 7

### EXPLICACIÓN:

A ambas variables se les asigna una representación binaria del número 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

### EXPLICACIÓN:

El constructor de Calculator recibe un número que es multiplicado por 10 y asignado a la variable de instancia num, por lo que imprime 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
- java.lang and util.random are redundant
- you dont need to import java.lang

### EXPLICACIÓN:

Tu no puedes omitir la java.util porque la clase Random pertenece a este paquete, y no compila al no encontrar la clase.

Los últimos enunciados no son correctos porque java.lang y util.random son diferentes paquetes, y en java.lang reside todas las clases fundamentales para el lenguaje de Java.

## 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

### EXPLICACIÓN:

A pesar de que en las dos impresiones se hace un incremento uno es un post es decir se aplicará después de la impresión, y en la última línea se realiza antes de la impresión.

### 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

#### EXPLICACIÓN:

En el ciclo for se toma hr como variable de control del ciclo, inicia con un valor a 11 hasta disminuir en 6 unidades quedando con un valor de 6, pero al alcanzar la condición  $6 > 6$  termina el ciclo, por lo que sg sólo se incrementa en 5 unidades.

### 14. ¿CUÁLES SON VERDADERAS?

- a) An ArrayList is mutable:
- b) An Array has a fixed size
- c) An array is mutable
- d) An array allows multiple dimensions
- e) An arrayList is ordered
- f) An array is ordered

#### EXPLICACIÓN:

Un ArrayList se puede cambiar su tamaño por lo que si es mutable mientras un array tiene un tamaño fijo desde su definición, por lo que no lo es; un array permite arreglos multidimensionales y ambos tienen un orden

## 15. RESULTADO

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

//Errores de compilacion, necesita un bool el while

### EXPLICACIÓN:

En un while se permiten expresiones booleanas o valores booleanos por lo que un número no lo podría compilar el código.

## 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, se debe encadenar el stream por que se consume

### EXPLICACIÓN:

17 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?

### EXPLICACIÓN:

Considerando que y es una variable de tipo “y” que guarda la referencia de un objeto de x que implementa y, podría funcionar, de otro modo no porque de y no se puede instanciar.

18 WHEN A BYTE IS ADDED TO A CHAR, WHAT IS THE TYPE OF THE RESULT?

Respuesta: int

#### EXPLICACIÓN:

Cualquier operación matemática que involucra tipos int o menores como char, byte o short resulta en un int.

#### 19 THE STANDART APPLICATION PROGRAMMMING INTERFACE FOR ACCESING DATABASES IN JAVA?

Respuesta: JDBC

#### EXPLICACIÓN:

La API de conectividad de bases de datos Java (JDBC) proporciona acceso universal a los datos desde el lenguaje de programación Java. Mediante la API de JDBC, puede acceder a prácticamente cualquier fuente de datos, desde bases de datos relacionales hasta hojas de cálculo y archivos planos. La tecnología JDBC también proporciona una base común sobre la que se pueden crear herramientas e interfaces alternativas.

La API de JDBC se compone de dos paquetes:

java.sql

javax.sql

Se obtiene automáticamente ambos paquetes cuando descarga Java Platform Standard Edition (Java SE) 8.

Para utilizar la API de JDBC con un sistema de gestión de bases de datos en particular, necesita un controlador basado en tecnología JDBC para mediar entre la tecnología JDBC y la base de datos. Según diversos factores, un controlador puede estar escrito exclusivamente en el lenguaje de programación Java o en una mezcla del lenguaje de programación Java y los métodos nativos de la interfaz nativa de Java (JNI). Para obtener un controlador JDBC para un sistema de gestión de bases de datos en particular, consulte API de acceso a datos JDBC.

#### 20 WHICH ONE OF THE FOLLOWING STATEMENTS IS TRUE ABOUT USING PACKAGES TO ORGANIZE YOUR CODE IN JAVA ?

Respuesta: Packages allow you to limit access to classes, methods, or data from classes outside the package.

#### 21 FORMAS CORRECTAS DE INICIALIZAR UN BOLEANO

#### EXPLICACIÓN:

boolean a = (3>6); boolean f = false;

#### 22 PREGUNTA

Pregunta repetida

## 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();
    }
}
```

### EXPLICACIÓN:

El método doSomething() lanza dos excepciones una de tipo unchecked y otra checked , por lo que se debe tomar una acción con la excepción IOException, ya sea con un try-catch o especificando un throws en el método main, en este caso se hace uso de los dos porque uno se hace cargo de la runtimeexception aunque no sea necesario, y el throw funciona para la excepción checked, también se puede adicionar un catch y haríamos uso de un multicatch, como se muestra a continuación:

```
import java.io.IOException;

public class YException {
    public static void main(String[] args) {
        try {
            doSomething();
        }catch (RuntimeException exception){
            System.out.println(exception);
        }catch (IOException exception){
            System.out.println(exception);
        }
    }
    static void doSomething() throws IOException {
        if (Math.random() > 0.5){

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

## 24 RESULTADO

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

Respuesta:

No compila.

## EXPLICACIÓN:

No compila porque los métodos en una interface son públicos y abstractos, en este caso aunque esta implementado, debe tener un modificador de acceso publico no puede disminuir.

## 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);
    }
}
```

## EXPLICACIÓN:

Tenemos una clase Spider que extiende de Arthropod e implementa su propio método printName cambiando el tipo de parámetro en vez de un double ahora recibe un int, por lo que al mandar a llamar los métodos responden de acuerdo al argumento, el primero invoca la función con un int por lo que imprime Spider y después usa el método con un double por lo que invoca el método padre e 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]);  
        }  
    }  
}
```

### EXPLICACIÓN:

Imprime tres veces Wed, porque con el ciclo foreach ejecutamos lo mismo tres veces que es crear una variable de referencia Days le asignamos el arreglo que compone la clase enum e imprimimos el valor en el índice 2.

## 27 PREGUNTA

```
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);  
    }  
}
```

### EXPLICACIÓN:

Imprime: true 20 false, en x se asigna el resultado de la operación con operador lógico utilizado para la manipulación de bits y devuelve true sólo si ambos valores booleanos son diferentes; de lo contrario, devuelve false. Dentro de la operación en x a z se le asigna el valor de false.

## 28 PREGUNTA

```
class InicializacionOrder {
    static {add(2);}
    static void add(int num){
        System.out.println(num+"");
    }
    InicializacionOrder(){add(5);}
    static {add(4);}
    {add(6);}
    static {new InicializacionOrder();}
    {add(8);}
    public static void main(String[] args) {}
}
```

### EXPLICACIÓN:

Imprime: 2 4 6 8 5, porque al inicializar una clase los bloques estáticos son los primeros en ejecutarse, se ejecutan en el orden en que están escritos y se ejecutan por una única vez. Dentro de un bloque estático se instancia un objeto por lo que ahora se ejecutan los bloques de instancia en el orden en que están escritos y al final se ejecuta lo que está en el constructor.

## 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 dont match");
        }else {
            System.out.println("They match");
        }
    }
}
```

### EXPLICACIÓN:

Imprime: "They dont match", porque en realidad son dos objetos diferentes message1 se encuentra en el pool de strings, y message2 es un objeto independiente



### 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");
    }
}
```

#### EXPLICACIÓN:

Se imprime: 1 2 4, pero no llega a imprimir el 5 ni el 6 ya que se lanza la excepción en el catch y no se hace nada con esa excepción.

### 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

#### EXPLICACIÓN:

El código no compila porque el método `executeUpdate` indica que lanza una EXCEPCIÓN:  
***int executeUpdate(String sql) throws SQLException***

Por lo que debería contener un ***throws SQLException*** el método main o un try catch.

### 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;
    }
}
```

EXPLICACIÓN:

### 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]);
        }
    }
}
```

### 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++;
            }
        }
    }
}
```

EXPLICACIÓN:

### 35 WHICH OF THE FOLLOWING STATEMENT ARE TRUE:

- a) string builder es generalmente más rápido que string buffer
- b) string buffer is threadsafe; stringbuilder is not

### 36 PREGUNTA

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

EXPLICACIÓN:

### 37 THE CATCH CLAUSE IS OF THE TYPE:

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

### 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

### 39 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; respuesta: s1.equalsIgnoreCase(s2)

EXPLICACIÓN:

## 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);  
        }  
    }  
}
```

EXPLICACIÓN:

## 42 PREGUNTA

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

EXPLICACIÓN:

### 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);
    }
}
```

EXPLICACIÓN:

### 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

EXPLICACIÓN:

### 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);
    }
}
```

EXPLICACIÓN:

#### 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]+"");
    }
}
```

EXPLICACIÓN:

#### 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();
    }
}
```

EXPLICACIÓN:

#### 48 PREGUNTA

```
class Test {  
    public static void main(String[] args) {  
        String s1 = "hello";  
        String s2 = new String ("hello");  
        s2=s2.intern();  
        System.out.println(s1==s2);  
    }  
}
```

#### 49 ¿CUÁL DE LAS SIGUIENTES CONSTRUCCIONES ES UN CICLO INFINITO WHILE?

- while(true);
- while(1==1){}

// Pregunta

```
class SampleClass{  
  
    public static void main(String[] args) {  
  
        AnotherSampleClass asc =new AnotherSampleClass ();  
  
        SampleClass sc = new SampleClass();  
  
        //sc = asc;  
  
        //TODO CODE  
  
    }  
  
}  
  
class AnotherSampleClass extends SampleClass {}  
  
// Respuesta: sc = asc;
```

#### 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
```

## 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
```

## 52 PREGUNTA

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

## 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
```

## 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
```

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
```

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
```

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
```

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
```

59 Pregunta

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

```

}

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

```

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 ...

```

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

```

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
```

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
```

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;
```

```
}
```

```
}
```

```
}
```

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

```

#### 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  
  
    */
```

#### 71 Pregunta

```
public class Main {  
  
    public static void main(String[] args) throws ParseException {  
  
        Queue<String> products = new 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);
```

```
});
```

```
}
```

```
}/**
```

```
*p1
```

```
* p1
```

```
*
```

```
* p2
```

```
* p3
```

```
*/
```

72 Pregunta

```
public class Main {
```

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

```
System.out.println(2+3+5);
```

```
System.out.println(""+2+3*5);
```

```
}
```

```
// Salida: 10 + 215
```