BANCO DE PREGUNTAS JAVA

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;
                }
            }
        }
    }
}
```

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: sobreescritura de método adecuada, por cambio de parámetro.
    private int lightsaber(long x) { return 0; } // Correcto: no se está 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; } // 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 numero de dientes, y a su vez se invoca el constructor con tres parámetros e igual se manda una valor definido para el ultimo 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?

```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();
    }
}
```

// 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?

```java
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 recibiéndolo como un arreglo, por lo que retorna 2 y es el valor impreso.

7. RESULTADO

```java
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

```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);
    }
}
```

// 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?

```java
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 numeró 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

EXPLICACIÓN:

El constructor de Calculator recibe un numero que es multiplicado por 10 y asignado a la variable de instancia num, por lo que imprime 20.

11. QUE ASEVERACIONES SON CORRECTAS

```java
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

```java
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 varbible 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 multidimentsionales y ambos tienen un orden

15. RESULTADO

```java
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 numeró no lo podría compilar el código.

16 RESULTADO

```java
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 encadernar 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 PROGRAMMMMING 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

```java
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 acontinuación:

```java
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 into, 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

```java
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) {}
}
```

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

```java
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.exeuteUpdate("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:

Lo que imprimé en consola: *the profits are 676000, 18330000, 27980000*
Por polimorfismo se puede acceder al mismo método, sin embargo al método más específico del objeto es decir a la clase del objeto no al de la referencia.

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]);
        }
    }
}
```

EXPLICACIÓN:

Saldrá una excepción tipo RuntimeException debido a que en el for intentará acceder al índice 4 el cual no existe ya que el índice en lo array comienza en 0 por lo que la condición en el ciclo for debe ser i < arr.length.

34 PREGUNTA

```java
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:

Imprime: beans,2 beans,3
egg,2 egg,3
ham,2 ham,3
juice,2 juice,3

Porque dentro del ciclo for hay un ciclo while con la condición de que la variable dish sea menor al tamaño del arreglo en este caso 4, por lo que no alcanza la iteración para imprimir 4, la variable dish a ser locar dentro del for se vuelve a inicializar en cada iteración del for.

35 WHICH OF THE FOLLOWING STATEMENT ARE TRUE:

a) string builder es generalmente más rápido qué string buffer
b) string buffer is threadsafe; stringbuildder is not

El inciso a) es correcto porque StringBuilder al no tener métodos syncronized se ejecuta más rápido.
El inciso b) es verdad porque al no ser synchronized los métodos métodos de StringBuilder significa que no es thread-safe.

36 PREGUNTA

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

EXPLICACIÓN:

Lo que observamos aquí es la sobreescritura del método equals, podemos observar que la firma es correcta, sin embargo, no cumple con el contrato de Java para la implementación del método equals, por las siguientes razones:

La principal es que en la documentación encontramos que: La implementación del método equals() devuelve true solo cuando ambas referencias apuntan al mismo objeto. Y lo que se observa en el código es que recibe un objeto sin verificar si es una instancia de la clase o bien manejar un excepción para devolver false en caso de un CastException, tampoco verifica si recibe un null para devolver false.

Una vez que se cumpliera todo lo anterior ya se podría hacer ese return.

37 THE CATCH CLAUSE IS OF THE TYPE:

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

EXPLICACIÓN:

De acuerdo con la documentación cada bloque catch es un controlador de excepciones que maneja el tipo de excepción indicado por su argumento. El tipo de argumento, ExceptionType, declara el tipo de excepción que el controlador puede manejar y debe ser el nombre de una clase que herede de la clase Throwable. El controlador puede hacer referencia a la excepción con el nombre. Sin embargo si usamos Throwable en un catch también va a atrapar los errores que son lanzados por la JVM.

La buena practica con los try/catch es manejar las excepciones por lo que debería ser un de tipo Exception o una subclase de ésta.

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

```java
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:

Devuelve Equal porque la condición del If es verdadera ya que el método solo evalúa los caracteres si considerar si están en mayúsculas o minúsculas.

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:

El método .compareTo(String anotherString) compara dos string lexicográficamente. El resultado es negativo si el String precede del otro String, es positivo de lo contrario, y es igual a 0 si son iguales.

Después Arrays.sort() recibe el arreglo y el comparador, en este caso en una expresión lambda que ordenará de forma ascendente, si se hiciera b.compareTo(a) lo haría de forma descendente. Por lo que en la consola se mostraran los string ordenados de forma ascendente.

42 PREGUNTA

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

EXPLICACIÓN:

Se produce una ArrayIndexOutOfBoundsException porque Java no soporte índices negativos.

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:

Imprimirá 0 ya que la clase sólo tiene el constructor por defecto.

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:

Hay dos variables locales en el método main, la cuales deben ser inicializadas como c2 se inicializa después de c1, no se puede compilar la clase.

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:

La operación entre diferente primitivos no dará problema, los operadores post y pre tienen prioridad en la operación, después la multiplicación por lo que la operación que se ejecutará es:

11+(20*30) >>> 11 + 600>>> Y el resultado final será: 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]+"");
    }
}
```

EXPLICACIÓN:

El código no compilará debido a que el varargs esta mal escrito debe ser el ultimo parámetro.

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:

El código compila sin problema e imprime: plant leaves plant leaves, debido a que primero imprimirá el constructor de las superclases, en este caso es Object, despues Plant que imprime Plant, depúes como en el constructor de Forest invoca el constructor que recibe un String imprimirá leaves, después crea un objeto Tree por lo que se vuelve a invocar en cadena los constructores Object, despues Plant imprime plant y por ultimo el string leaves.

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);
    }
}
```

EXPLICACIÓN:

Devuelve verdadero porque al invocar intern() se toma el string que se encuentra en el pool de Srtrings, por lo que s1 y s2 señalan al mismo objeto.

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

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

EXPLICACIÓN:

Ambos serían infinitos al menos que se use un break dentro de ellos.

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);
    }
}
```

EXPLICACIÓN:

Ninguna de las condiciones se cumple por lo que nunca cambia el valor de z por lo que s e imprime 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";
    }
}
```

EXPLICACIÓN:

El constructor se llama infinitamente ya que es recursivo, por lo que se produce una Excepción StackOverflowError.

52 PREGUNTA

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

EXPLICACIÓN:

No compilará porque no esta inicializada la variable a.

53 PREGUNTA

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

EXPLICACIÓN:

La primer línea realiza la operación e imprime 10, y en la segunda línea al tener un string al inicio imprime +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");
    }
}
```

EXPLICACIÓN:

Todos los if's se ejecutan, y las condiciones que se cumples es la primera, así que se imprime Here1; y la tercera e imprime 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;}
}
```

EXPLICACIÓN:

El método count no puede ser utilizado ya que es un método de instancia, si es estico se podría usar de esa manera, u otra opción es creando una instancia de Main o de 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");
    }
}
```

EXPLICACIÓN:

Se imprime magic town, porque es el método main que se reconoce como entrada para ejecutar la aplicación.

57 PREGUNTA

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

EXPLICACIÓN:

Se imprime 2, porque se ejecuta una operación y el post-increment sucede después, por lo que la siguiente línea 1.

58 PREGUNTA

```java
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);
    }
}
```

EXPLICACIÓN:

No compila debido a que no esta especificado el tipo de elementos en el arraylist. Para que compile podría quitarse <E>, o especificar que va recibir Integers.

59 PREGUNTA

```java
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);
    }
}
```

EXPLICACIÓN:

No compilara ya que break es una palabra reservada y no puede ser usada.

60 PREGUNTA

```java
public 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);
    }
}
```

EXPLICACIÓN:

La clase App tiene una sobrecarga de constructores, por lo que en este caso en la primer línea se invoca el constructor que recibe integers y en el segundo se invoca el constructor que recibe un objeto por lo que imprimirá 3, y la siguiente 4.

61 PREGUNTA

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

EXPLICACIÓN:

Imprimirá everething, porque el operador ternario primero evalua si 42 es menor a 40 al ser falso, se evalua si 42 es mayor a 50 que también es falso por loque everething se asigna a s.

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));
        }
    }

}
```

EXPLICACIÓN:

Se imprime 333.6789 234.111 porque NumberFormat no hace nada con parse. Y se debe añadir un throws Exception al método main.

63 PREGUNTA

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

EXPLICACIÓN:

Devuelve true porque corresponden al mismo objeto en el pool de string en s2 a pesar de tener un concatenación no se le añade nada.

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;
        }
    }
}
```

EXPLICACIÓN:

Las etiquetas en los case no pueden ser de tipo booleano y deben de constantes en tiempo de compilación, por lo que el código no compilará.

65 PREGUNTA

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

EXPLICACIÓN:

Se imprimé -100 solo se le asigna el menos para hacer lo como un valor negativo.

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");
        }
    }
}
```

EXPLICACIÓN:

No compila problema ya que las etiquetas de los case deben ser del mismo tipo de la opción que van revisar en este caso 200 ya no es de tipo byte es int.

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");
    }
}
```

EXPLICACIÓN:

Se produce un ClassCastException debido a que no se pude hacer un cast de un objeto de tipo de la superclase a uno de la subclase.

68 PREGUNTA

```java
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;
        }
    }
}
```

EXPLICACIÓN:

Al no tener un break, y encontrar la coincidencia se ANY FRUIT WILL DO APPLE MANGO BANANA.

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());
    }
}
```

EXPLICACIÓN:

A la clase Animal le hace falta un constructor sin argumentos, para que el constructor en Dog Dog(String breed) funcione y compile.

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));
        }
    }
}
```

EXPLICACIÓN:

Imprime: 333.6789 234.111, porque NumberFormat no hace nada con parse().

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);
        });
    }
}
```

EXPLICACIÓN:

Imprime: p1 p1 p2 p3; El método peek() Recupera, pero no elimina, el encabezado de la cola representada por esta deque, o devuelve null si esta deque está vacía. El método poll() Recupera y elimina la cabecera de la cola representada por esta deque (en otras palabras, el primer elemento de esta deque), o devuelve null si esta deque está vacía.

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);
    }
}
```

EXPLICACIÓN:

Imprime: 10 +215; en el primer caso ejecuta la operación y en la segunda línea al comenzar con un string comienza un concatenación pasando a ser string los números.