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;
            }
        }
    }
}
```

//1 FINN 2 FINN 2 JAKE 3 FINN 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;} //El modificador de acceso en la
subclase no puede ser más restrictivo que el modificador de acceso en la clase
base.
    protected int lightsaber (long x){return 0;} //Correcto, no se está
sobreescribiendo el método ya que la signature del método es diferente
    private int lightsaber (long x){return 0;} //Correcto, no se está
sobreescribiendo el método ya que la signature del método es diferente
    protected long lightsaber (int x){return 0;} // Error, al tener la misma
signature debe de tener el mismo 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 metodo
}
```

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();
    }
```

El constructor que se llama en el main llama al constructor que recibe dos parámetros que a su vez llama al constructor de tres parámetros dando como salida 15, 16, 6.

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();
    }
}
```

Al crear un nuevo Spider llamamos al constructor sin argumentos de la superclase seguido del constructor sin argumentos de Spider, el método run() modifica la variable type que fue heredada de Arachnid, por lo que la salida es arachnid spider s s

5. Resultado

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

El decremento es prefijo en la primera salida por lo que la salida será:

2

2

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

El decremento es prefijo en la primera salida por lo que la salida será:

997

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";
    }
}
```

Los primeros dos métodos que se mandan a llamar están definidos en la clase, para el método que tiene tres parámetros podemos usar el método *String... s* ya que indica que puede recibir una cantidad variable de parámetros de tipo String Salida: 1, 4, 2

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();
    }
}
```

El programa no compila ya que lanza ClassCastException porque se está intentando castear el objeto de una clase con otro con el que no se tiene una relación de herencia.

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 porque se está asignando un int a una String.

9. 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);
    }
}
```

El código imprime dos veces el número 7 alocado en number1

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();
    }
}
```

El código imprime 20 ya que con el método calc(2) se modifica la variable num.

11. ¿Qué aseveraciones son correctas?

```
class ImportExample {
    public static void main (String [] args){
        Random r = new Random();
```

```
        System.out.println(r.nextInt(10));
    }
}
```

*No se puede omitir import java.util.Random; para que el código compile.

*El compilador java importa el package java.lang; por default

12. Resultado

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

El código imprime 10 ya que el incremento de var es postfijo, la segunda salida imprime 12 ya que se modifica por el incremento postfijo anterior y el incremento prefijo de la misma variable.

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);
    }
}
```

La expresión hr-= 1 también se puede representar como hr--, al término del for la variable hr decrementa en 1 cinco veces, la variable sg aumenta en 1 cinco veces por lo que la salida es sg =5

14. ¿Qué aseveraciones son verdaderas?

* Un ArrayList es mutable y ordenado

* Un array tiene tamaño fijo, es mutable, permite múltiples dimensiones y es ordenado

15. Resultado

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

Se presenta un error de compilación ya que la condición del while no es de tipo bool.

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));
    }
}
```

Se presenta una Exception at runtime, el error "java.lang.IllegalStateException: stream has already been operated upon or closed" se produce cuando intenta reutilizar un Stream después de que ya se haya consumido o cerrado. Esto se debe a que los Streams en Java 8 están diseñados para usarse solo una vez.

18 Cuando un **byte** se suma con un **char** el resultado es de tipo **int**

19 La API para accesar a bases de datos en java es JDBC

20 Sentencias para organizar el código en Java

Los packages permiten limitan el acceso a clases, métodos o datos a otras clases fuera del package.

21 Pregunta

Forma correcta de inicializar un valor boolean:

* boolean a = (3>6);

23 Pregunta

```
class Test{
    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();
    }
}
```

*

La excepción lanzada en el método doSomething() no coincide con la que se indica en la claúsula **throws IOException** que difiere de **RuntimeException** que es una unchecked exception**.**

24 Resultado

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

Debido a que el método de la interfaz es public implícitamente no podemos reducir la visibilidad en la sobreescritura del método en la clase Manager.

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);
    }
}
```

La salida al llamar al método printName() con un entero como parámetro imprime Spider, la salida al llamr al método printName() con un double como parámetro 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]);
        }
    }
}
```

En el for solo se hace una iteración ya que solo se tiene un objeto de tipo Days, se asigna el arreglo a d2 que imprime 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);
    }
}
```

El operador ^ regresa verdadero si las operaciones booleanas son diferentes, por lo que x = true = true ^ 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) {}
}
```

El compilador ejecuta primero las sentencias static en el orden en el que están definidas, en segundo lugar, ejecuta los métodos static y por último los inicializadores de instancia 2 4 6 8 5

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");
        }
    }
}
```

message1 y message2 son objetos diferentes aunque el contenido sea el mismo, por lo que la evaluación **message1!=message2** es verdadera They dont match

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");
    }
}
```

El código compila, la excepción que lanza el método toString() es manejada por el catch que en su body lanza la misma excepción, al estar en un catch no es posible que se maneje // Salida 1 2 4 NullPointerException

```
31 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

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;
    }
}
```
// the profits are 676000, 18330000, 27980000

33

```
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]);
        }
    }
}
```
DUHAST
FEEL
YELLOW
FIX YOU
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4
    at Main.main(Main.java:5)

35 StringBuilder es generalmente más rápido qué StringBuffer

StringBuffer es threadsafe, StringBuildder no lo es.

36

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

38 Un bucle for mejorado también llamado for each, ofrece una sintaxis simple para iterar a través de una colección pero no puede ser utilizado para eliminar elementos de una colección

40

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

El método equalsIgnoreCase() compara carácter por carácter sin importar si son mayúsculas o minúsculas.

Salida: Equal;

```
41 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);
        }
    }
}
```

/* apple

banana

grapes

pears */

42

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

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException debido al index -1 en el arreglo countsofMoose

43

```
class Salmon{
    int count;
    public void Salmon (){
        count = 4;
    }
    public static void main(String[] args) {
        Salmon s = new Salmon();
        System.out.println(s.count);
    }
}
```

Aunque lo parezca, el constructor sin argumentos no está definido, por lo que no se manda a llamar al método Salmon() por lo que no se modifica count Salida: 0 -> cero

44

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

Imprime 0

45

```
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)

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

No compila, el argumento *String…* y debe ser el último en la lista de parámetros

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*/

48

```
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 ala
cadena
        System.out.println(s1==s2);
    }
}
```

// Salida: true

49 Las siguientes sentencias son while infinitos:

* while(true);

* while(1==1){}

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

Respuesta: sc = asc;

50

```
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 debido a que es una función recursiva que no tiene una apropiada condición de término.

52 Pregunta

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

La cadena **String a** no se inicializa por lo que no se puede llamar al método toString()

53

```java
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

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

// salida: Here1 , here 3

55 Pregunta

```java
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;}
}
```

No compila porque se manda a llamar a un método de instancia sin haberla creado

56

```
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 ...

```
class App {
        public static void main(String[] args) {
            int i=42;
```

```
            String s = (i<40)?"life":(i>50)?"universe":"everything";
            System.out.println(s);
        }
    }
```

// Salida: everything

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

```
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 compilación, se está asignando un bool al switch de tipo int

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 compilación, se está asignando un int al switch de tipo 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;
        }
    }
}
```

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());
    }
}
```

// Error de compilación porque el constructor sin argumentos de la superclase no está definido

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