GUÍA APX Semana 5

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;
			}
		}
	}
	}
}//Salida: 1 FINN 2 FINN 2 JAKE 3 FINN 3 JAKE
```

Explicación:

El código imprime la combinación del valor de x y cada elemento de la matriz at. El bucle externo controla los valores de x, y el bucle interno recorre los elementos de la matriz. En la primera iteración de x, se imprime solo "1 FINN" porque el break detiene el bucle interno al encontrar que x == 1. En las siguientes iteraciones (x == 2 y x == 3), el break ya no se ejecuta, por lo que imprime todos los elementos de la matriz para esos valores de x. Por eso, el resultado es "1 FINN", "2 FINN", "2 JAKE", "3 FINN" y "3 JAKE".

2. ¿Que 5 lineas son correctas?

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

class Saber extends Light{
	// Insertar los métodos válidos en esta subclase
}
```

a) `private int lightsaber (int x){return 0;}` // Es incorrecta porque no se puede reducir la visibilidad de un método heredado.

b) `protected int lightsaber (long x){return 0;}` // Es una sobrecarga válida del método porque cambia el tipo de parámetro de int a long.

c) `private int lightsaber (long x){return 0;}` // Es una sobrecarga válida, ya que cambia el tipo de parámetro. Al ser un método nuevo, puede tener un modificador de acceso privado.

d) `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.

e) `protected long lightsaber (int x, int y){return 0;}` // Es una sobrecarga válida porque añade un segundo parámetro.

f) `public int lightsaber (int x){return 0;}` // Es válido, ya que mantiene la misma firma del método original y aumenta la visibilidad de protected a public.

g) `protected long lightsaber (long x){return 0;}` //Válido, porque cambia tanto el tipo de parámetro como el tipo de retorno.

3. ¿Que resultado arroja?

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

	public Mouse (int weight){
	this(weight,16);
	}
	public Mouse (int weight, int numTeeth){
		this(weight, numTeeth, 6);
	}
	public Mouse (int weight, int numTeeth, int numWhiskers){
		this.weight = weight;
		this.numTeeth= numTeeth;
		this.numWhiskers = numWhiskers;
	}
	public void print (){
		System.out.println(weight + ""+ numTeeth+ ""+ numWhiskers);
	}
	public static void main (String [] args){
		Mouse mouse = new Mouse (15);
		mouse.print();
	}
} // Salida: 15 , 16 , 6
```

Explicación:

Se define una clase Mouse con tres constructores sobrecargados que permiten inicializar los atributos weight, numTeeth y numWhiskers de diferentes maneras.

Cuando se crea un objeto Mouse con el valor 15 en new Mouse(15);, se llama al primer constructor, que a su vez llama al segundo constructor pasando 15 y 16 como parámetros. Luego, el segundo constructor llama al tercer constructor con los valores 15, 16, y 6. Finalmente, los valores se asignan a los atributos de la clase y se imprime la salida: 15 16 6.

4. ¿Cual es la salida?

```
class Arachnid {
public String type = "a";
	public Arachnid(){
		System.out.println("arachnid");
	}
}
class Spider extends Arachnid{
	public Spider(){
		System.out.println("spider");
	}
	void run(){
		type = "s";
		System.out.println(this.type + " " + super.type);
	}
	public static void main(String[] args) {
		new Spider().run();
	}
}
// Salida: arachnid spider s s
```

Explicación:

Al ejecutar el código, primero se crea una instancia de Spider, lo que hace que se llame de primero y al constructor de la clase padre Arachnid, imprimiendo "arachnid". Luego, se ejecuta el constructor de Spider, que imprime "spider".

Después, el método run() se ejecuta, cambia el valor de type a "s" y lo imprime dos veces, ya que this.type y super.type se refieren al mismo valor.

5. Resultado

```
class Test {
	public static void main(String[] args) {
	int b = 4;
	b--;
	System.out.println(--b);
	System.out.println(b);
	}
}
class Sheep {
public static void main(String[] args) {
	int ov = 999;
	ov--;
	System.out.println(--ov);
	System.out.println(ov);
	}
} // Respuesta correcta: 997, 997
```

Explicación:

Se comienza con la variable ov inicializada a 999. Primero, se reduce en 1 con ov--, dejando el valor en 998. Luego, se reduce nuevamente en 1 con --ov antes de imprimir, por lo que se imprime 997. Finalmente, se vuelve a imprimir el valor actual de ov, que sigue siendo 997.

6. Resultado

```
class Overloading {
public static void main(String[] args) {
	System.out.println(overload("a"));
	System.out.println(overload("a", "b"));
	System.out.println(overload("a", "b", "c"));
}
public static String overload(String s){
	return "1";
}
public static String overload(String... s){
	return "2";
}
public static String overload(Object o){
	return "3";
}
public static String overload(String s, String t){
	return "4";
}
} // Salida: 1, 4, 2
```

Explicación:

Se llaman diferentes métodos sobrecargados con distintos números de argumentos:

1. overload("a") usa el método que recibe un solo String, por lo que imprime 1.
2. overload("a", "b") utiliza el método que recibe dos String, así que imprime 4.
3. overload("a", "b", "c") usa el método que acepta un número variable de String, imprimiendo 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();
	}
```

```
}
```

Explicación:
Se crea un objeto de Base1, pero se intenta convertir a Base2, lo cual no es posible porque Base1 y Base2 son clases independientes sin una relación de herencia padre-hijo. Esto causa una excepción ClassCastException en tiempo de ejecución.

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 codigo no compila
```

Explicación:
El código no compila porque se intenta asignar un valor int (numFish + 1) a una variable String sin convertirlo primero. Primero se requiere hacer la conversión explícita para realizar esta asignación.

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);
	}
}
//Salida: 7 7 ojo que imprime dos veces number 1
```

Explicación:
En este ejercicio, number1 se define en formato binario como 0b0111, que es 7 en decimal. number2 se define en formato octal como 0111_000, que se convierte a 4672 en decimal. Pero en el código se imprime dos veces la variable 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();
	}
} // Salida: 20
```

Explicación:
El código crea un objeto Calculator, actualiza num a 20 mediante el método calc, y luego 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));
	}
}
```

a) If you omit java.util import statements java compiles gives you an error // Se necesita importar java.util.Random; de lo contrario, se produce un error de compilación.
b) java.lang and util.random are redundant // no son redundantes; pertenecen a paquetes diferentes.
c) you dont need to import java.lang // No es necesario importar java.lang porque se importa automáticamente.

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:
El código imprime 10 primero, ya que el operador var++ muestra el valor antes de incrementarlo. Luego, imprime 12, ya que ++var incrementa el valor antes de imprimirlo.

13. Resultado

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

```
// Salida sg=5; Respuesta correcta mn = 11
```

Explicación:
El código inicializa hr en 11 y decrementa su valor en cada iteración del bucle hasta que hr es 6. En cada iteración, sg se incrementa en 1. Después de que el bucle termina, sg cuenta cuántas veces se ejecutó el bucle, resultando en 5. Por lo tanto, el resultado impreso es sg=5.

14. Cuales son verdad

A. An ArrayList is mutable. //Verdadero. Se pueden agregar, eliminar o modificar elementos en un ArrayList.
B. An Array has a fixed size // Verdadero. El tamaño de un array no cambia una vez creado.
C. An array is mutable // Incorrecto. Aunque los elementos pueden cambiar, el tamaño del array no lo hace.
D. An array allows multiple dimensions // Incorrecto. Aunque los arrays pueden ser multidimensionales, la afirmación puede ser confusa en términos de mutabilidad.
E. An arrayList is ordered // Verdadero. Los elementos de un array están en un orden específico basado en sus índices.
F. An array is ordered // Verdadero. En un ArrayList, los elementos mantienen el orden en que fueron añadidos.

15. Resultado

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

Explicación:
El código intenta usar --negotiate en la condición del bucle while, lo cual no es válido porque el resultado de --negotiate es un valor entero, no un booleano. La condición de un bucle while debe ser un booleano (true o false).

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 encadernar el stream por que se
consume
```

Explicación:

El código falla porque el flujo se consume al usarlo para filtrar y luego intentar imprimir. Para solucionarlo, se deben encadenar las operaciones de filtro e impresión en una sola línea, evitando el error.

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

Explicación:
La asignación Object x = y es siempre legal, porque Object es la superclase de todas las clases en Java. No importa qué tipo específico tenga y, siempre se puede asignar a una variable de tipo Object.

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

Explicación:
Los operandos de tipo byte, short o char se promocionan a int en las operaciones aritméticas. Esto significa que, antes de realizar la suma, tanto el byte como el char se convierten a int. Por lo tanto, la operación se realiza utilizando aritmética int, y el resultado de la suma será de tipo int.

19 Pregunta
The standart application programmmming interface for accesing databases in java?

* JDBC segun CHATGPT

Explicación:
La API estándar para acceder a bases de datos en Java es JDBC (Java Database Connectivity). JDBC proporciona un conjunto de clases e interfaces que permiten a las aplicaciones Java conectarse e interactuar con bases de datos relacionales.

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

Explicación:
Los paquetes permiten limitar el acceso a clases, métodos o datos desde clases que estén fuera del paquete. Esto ayuda a controlar la visibilidad y el acceso a los elementos del código al organizarlos en paquetes.

21 Pregunta
Forma correcta de inicializar un boleano
* boolean a = (3>6);

Explicación:
La expresión boolean a = (3 > 6); es válida porque 3 > 6 evalúa a false, que es un valor booleano. Por lo tanto, a se inicializa con 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();
	}
}
* Adding throws IOException to the main() method signature
```

Explicación:
El código tiene un error de compilación porque el método main() no maneja la excepción IOException que puede lanzar el método doSomething(). Para corregir esto, se debe declarar que el método main() puede lanzar IOException, permitiendo así que el código compile y funcione correctamente.

24 Resultado

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

Explicación:
El método interviewConducted en la interfaz Interviewer es implícitamente public. En la clase Manager, el método interviewConducted debe ser declarado como public para cumplir con la implementación de la interfaz.

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);
	}
} // Spider, Arth
```

Explicación:
En el código, cuando se llama a printName(4), se imprime "Spider" porque el método que acepta un int está en la clase Spider. Cuando se llama a printName(9.0), se imprime "Arth" porque no hay un método que acepte un double en Spider, por lo que se usa el de la clase padre Arthropod.

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]);
		}
	}
} // Wed Wed Wed
```

Explicación:

Hay un bucle que itera sobre cada valor del enum Days. Durante cada iteración, se crea un nuevo arreglo d2 que contiene los valores del enum, y siempre se imprime el tercer valor del arreglo, que es "Wed". Como el bucle recorre tres veces (una por cada día en el enum), el resultado final es que "Wed" se imprime tres veces.

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);
	}
}// true 20 false
```

Explicación:

El código verifica si y es distinto de 10, asigna false a z, y realiza una operación XOR. Como resultado, x es true, y sigue siendo 20, y z se convierte en false. La salida es true 20 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) {}
} //2 4 6 8 5
```

Explicación:
Cuando la clase InitializacionOrder se carga, se ejecutan primero los bloques estáticos, imprimiendo 2 y luego 4. Luego, se crea una instancia de la clase, lo que activa los bloques de instancia y el constructor. Durante esta creación de la instancia, se imprimen 6, 8, y finalmente 5 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");
		}
	}
}// They dont match
```

Explicación:

En este código, message1 y message2 contienen el mismo texto, pero message1 es una cadena literal y message2 se crea con el operador new String(), lo que significa que son dos objetos diferentes en memoria. La comparación message1 != message2 verifica si las referencias a los objetos son distintas, lo que es cierto en este caso. Por lo tanto, se imprime 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");
      }
} // Salida 1 2 4 NullPointerException
```

Explicación:
En el código, el método run imprime 1 y 2, luego intenta llamar a name.toString(), pero como name es null, se lanza una NullPointerException. Esta excepción es capturada en el bloque catch, que imprime 4 y luego relanza la excepción. Debido a que la excepción es relanzada, el flujo se interrumpe, por lo que no se ejecuta System.out.println("6") en el método main.

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 intenta conectar a una base de datos y realizar una inserción en la tabla User. Si la conexión es exitosa, se inserta una fila con el ID 500 y el nombre 'Ramesh'. Luego, imprime el número de filas afectadas por la inserción, que normalmente será 1 si la operación se realiza correctamente.

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
```

Explicación:
El código crea tres objetos: uno de la clase MarvelClass y dos de sus subclases, MarvelMovieA y MarvelMovieB. Luego, se llama al método getHash() en cada objeto. El método en MarvelClass devuelve 676000, en MarvelMovieA devuelve 18330000, y en MarvelMovieB devuelve 27980000.

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]);
			}
	}
} //4 An arrayindexoutofbondsexception
```

Explicación:

El código intenta imprimir cada elemento de un array de cadenas. Sin embargo, el bucle se ejecuta hasta i <= arr.length, lo que lleva a un error cuando i alcanza el valor de 4, que está fuera del rango válido de índices del array (0 a 3). La salida es la siguiente:

DUHAST
FEEL
YELLOW
FIX YOU
ArrayIndexOutOfBoundsException

34 pregunta

```
class Menu {
	public static void main(String[] args) {
	String[] breakfast = {"beans", "egg", "ham", "juice"};
```

```
            for (String rs : breakfast) {
                  int dish = 2;
                  while (dish < breakfast.length) {
                        System.out.println(rs + "," + dish);
                        dish++;
                  }
            }
      }
} * Respuesta correcta: ONCE */
```

Explicación:
El código imprime cada elemento del array breakfast junto con los números 2 y 3. Para cada elemento del array, el bucle while imprime el elemento actual y los valores 2 y 3. La salida será la siguiente:

```
Salida:
beans,2
beans,3
egg,2
egg,3
ham,2
ham,3
juice,2
juice,3
```

35 pregunta
Which of the following statement are true:
* string builder es generalmente más rápido qué string buffer
* string buffer is threadsafe; stringbuildder is not

Explicación:

- StringBuilder es más rápido porque no tiene que hacer un trabajo extra para asegurarse de que múltiples hilos no interfieran entre sí.
- StringBuffer puede ser usado de forma segura cuando varios hilos están trabajando con él al mismo tiempo porque se asegura de que los cambios no se mezclen. StringBuilder, en cambio, no tiene esa protección, así que es más rápido, pero solo debe usarse en un solo hilo.

36 pregunta

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

Explicación:
El código no compila porque el método equals realiza un cast sin verificación, lo que puede llevar a errores en tiempo de ejecución. Además, el uso de == para comparar Integer es menos seguro que usar equals().

37 pregunta
The catch clause is of the type:

A. Throwable //
B. Exception but NOT including RuntimeException
C. CheckedException
D. RunTimeException
E. Error

Explicación:

Capturan las excepciones comprobadas (checked exceptions) pero no las excepciones de tiempo de ejecución (unchecked exceptions) como RuntimeException. Esto se debe a que el tipo de catch más general es Exception, pero para capturar específicamente las excepciones que no son de tiempo de ejecución (como RuntimeException), se debe utilizar Exception excluyendo RuntimeException.

38 pregunta
An enhanced for loop
* also called for each, offers simple syntax to iterate through a collection but it can´t be used to delete elements of a collection

Explicación:
El bucle mejorado (o "for-each") en Java simplifica la iteración sobre una colección, pero no permite eliminar elementos durante la iteración. Intentar hacerlo causará una excepción ConcurrentModificationException.

39 pregunta
Which of the following methods may appear in class Y, which extends x ?
public void doSomething(int a, int b){…}

Explicación:
En la clase Y, que extiende X, el método public void doSomething(int a, int b) puede aparecer si mantiene la misma visibilidad (public), firma (mismo nombre y parámetros) y excepciones que el método en X.

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:
El programa imprimirá "Equal" porque equalsIgnoreCase considera que "Java" y "java" son iguales al ignorar las diferencias de mayúsculas y 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);
		}
	}
}
/* Salida: apple
banana
grapes
pears */
```

Explicación:
El código ordena un arreglo de frutas en orden alfabético usando Arrays.sort con una expresión lambda que compara las frutas mediante compareTo (Comparable). Luego, imprime cada fruta en el nuevo orden alfabético. El resultado es una lista de frutas ordenadas: "apple", "banana", "grapes", "pears".

42 pregunta

```
public class Main {
	public static void main(String[] args) {
		int[]countsofMoose = new int [3];
		System.out.println(countsofMoose[-1]);
	}
}
//this code wull trow an arrayindexoutofboundsexpression
```

Explicación:
El programa lanzará una excepción ArrayIndexOutOfBoundsException en tiempo de ejecución porque los índices negativos no son válidos para los arreglos en Java.

43 Pregunta

```
class Salmon{
	int count;
```

```
	public void Salmon (){
		count =4;
	}
	public static void main(String[] args) {
		Salmon s = new Salmon();
		System.out.println(s.count);
	}
} // Salida: 0 -> cero
```

Explicación:
El código intenta definir un constructor con el nombre Salmon, pero como tiene un tipo de retorno (void), no se considera un constructor. Por lo tanto, el campo count no se inicializa a 4 y permanece en su valor predeterminado 0. El programa imprimirá 0.

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:
El código no compila porque usa las variables c1 y c2 antes de declararlas en el método main. Además, la variable estática v se imprime en runlap antes de ser inicializada, lo que no causa un error de compilación pero podría resultar en un valor predeterminado.

45 pregunta

```
class Foo {
	public static void main(String[] args) {
		int a=10;
		long b=20;
		short c=30;
		System.out.println(++a + b++ *c);
	}
} // salida: 611 (11+20*30)
```

Explicación:
El código imprime 611 porque incrementa a a 11, multiplica el valor de b (20) por c (30), y luego suma el resultado (600) a a (11).

46 pregunta

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

```
		new Shop().go("welcome",1);
		new Shop().go("welcome", "to", 2);
	}
	public void go (String... y, int x){
		System.out.print(y[y.length-1]+"");
	}
}  // Compilation fails
```

Explicación:
El código no compila porque los parámetros varargs (String... y) deben ser el último en la lista de parámetros. El método debe ser corregido para que int x esté antes de String... y.

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();
	}
}
/*SALIDA: plant
leaves
plant
leaves*/
```

Explicación:
Al crear una instancia de Forest, se llama al constructor de Tree (super("leaves")), que imprime "plant" (constructor de Plant) y luego "leaves" (constructor de Tree).
Luego, se crea otra instancia de Tree dentro del constructor de Forest, que nuevamente imprime "plant" y "leaves".

48 Pregunta

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

```
            System.out.println(s1==s2);
      }
} // Salida: true
```

Explicación:
El código imprimirá true porque s2.intern() hace que s2 apunte al mismo objeto de cadena en el pool de cadenas que s1. Por lo tanto, s1 y s2 son referencias al mismo objeto.

49 pregunta
Cuál de las siguientes construcciones es un ciclo infinito while:
* while(true); // Este código entra en un ciclo infinito. El punto y coma ; al final indica que el cuerpo del ciclo está vacío.
* while(1==1){} //Este código también crea un ciclo infinito. La condición 1==1 siempre es verdadera, y el cuerpo del ciclo es un bloque vacío {}.

```
// 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;
```

Explicación:
La asignación sc = asc; es válida en este contexto porque AnotherSampleClass es una subclase de SampleClass, permitiendo que una instancia de AnotherSampleClass sea asignada a una variable de tipo SampleClass.

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
```

Explicación:
El código imprimirá 0 porque ninguna de las condiciones en el if se cumple, así que z permanece en su valor inicial 0.

51 Pregunta

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

Explicación:
El código lanzará un StackOverflowError debido a la recursión infinita en el constructor de la clase course. Cada instancia de course crea otra instancia de course, causando un desbordamiento del stack.

52 Pregunta

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

Explicación:
El código lanza un error de compilación porque intenta utilizar la variable a antes de que haya sido inicializada. Para corregir el error, a debe ser inicializada antes de su uso.

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
```

Explicación:
El programa imprime 10 en la primera línea debido a que realiza una suma aritmética de 2 + 3 + 5. En la segunda línea, imprime +235, ya que concatena la cadena "+" con 2, 3 y 5, resultando en la cadena "+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
```

Explicación:
El programa imprime "Here1" porque a es igual a b, y "Here3" porque a es mayor o igual que b. No imprime nada para la segunda condición porque a no es diferente de b.

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
```

Explicación:
El código no compila porque el método count es de instancia y se está llamando de manera incorrecta. Para corregirlo, el método debe ser estático o debe ser llamado en una instancia de la clase count

56 Pregunta

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

Explicación:
El programa imprime magic town porque solo el método public static void main(String[] args) es el punto de entrada reconocido por la JVM. Los otros métodos main y mina no afectan la salida.

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
```

Explicación:
El programa imprime 2 en la primera línea porque a++ usa el valor actual de a (que es 0) antes de incrementarlo, sumándolo con 2. Luego, imprime 1 en la segunda línea, que es el nuevo valor de a después del incremento.

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
```

Explicación:
El código no compila porque E no está definido. Para solucionarlo, se debe usar un tipo específico, como Integer, en lugar de E.

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
```

Explicación:
El códigol no compila porque usa break, que es una palabra reservada en Java, como nombre de un método. Esto causa un error de sintaxis.

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 …
```

Explicación:

En el código proporcionado, se crean dos instancias de App con diferentes argumentos.

Para la primera instancia new App(100), el argumento 100 se convierte automáticamente en un Integer, por lo que se llama al constructor que acepta un Integer, que imprime "3".

Para la segunda instancia new App(100L), el argumento 100L es un long que se convierte en Long, el cual es un tipo Object. Por lo tanto, se utiliza el constructor que acepta un Object, que imprime "4".

El constructor sin parámetros (App()) no se utiliza en este caso.

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
```

Explicación:
El código evalúa dos condiciones ternarias. Dado que i es 42, que no es menor que 40 ni mayor que 50, el valor de s se asigna como "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
```

Explicación:
El error en el código proporcionado está relacionado con la excepción ParseException que puede ser lanzada por el método parse de NumberFormat. Este método puede lanzar una ParseException, la cual debe ser manejada con un bloque try-catch o declarada en la firma del método con throws.

63 Pregunta

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

Explicación:
El código compara dos cadenas, s1 y s2, que ambas contienen "OCAJP". Debido a que "OCAJP" se almacena en el pool de cadenas y la concatenación con una cadena vacía no

crea una nueva referencia, s1 y s2 apuntan al mismo objeto. Por lo tanto, s1 == s2 es verdadero y el programa imprime 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
```

Explicación:
El código da un error de compilación porque en un switch, los casos (case) deben ser constantes o valores literales. Las expresiones booleanas como score < 70 y score >= 70 no son válidas en este contexto. Además, el default debería estar al final o seguido por un break.

65 Pregunta

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

Explicación:
El código imprime -100 porque el operador ++ incrementa el valor de a después de usarlo, y el operador unario negativo convierte 100 en -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
```

Explicación:
El código da un error de compilación porque los valores en los casos del switch se tratan como enteros (int), y hay una posible pérdida de conversión al tipo byte. El compilador espera constantes byte para evitar problemas de conversión.

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
```

Explicación:

El programa lanza una excepción ClassCastException debido al intento de convertir un objeto de A a B cuando en realidad no es una instancia de B. El casting no es válido porque obj1 es una instancia de A, no de B. El casting solo es válido si el objeto es realmente una instancia de la clase o subclase a la que se quiere convertir.

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;
                }
        }
}
```

Explicación:
El código imprimirá ANY FRUIT WILL DO APPLE MANGO BANANA. Esto ocurre porque, al no haber coincidencia con el valor "mango", se ejecuta el bloque default y luego sigue ejecutando todos los casos subsiguientes debido a la falta de 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
```

Explicación:

El código tiene un problema porque el constructor de `Dog` que recibe solo `breed` no llama al constructor de Animal, por lo que name no se inicializa y es null.

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
*/
```

Explicación:
El código utiliza NumberFormat para configurar el formato de números con un máximo de 2 dígitos fraccionarios. Sin embargo, al usar nf.parse(s), los valores se convierten a números sin aplicar el redondeo o truncamiento especificado. Por lo tanto, se imprime el número original de cada cadena, mostrando todos los dígitos fraccionarios sin modificar.

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
*/
```

Explicación:
El código crea una cola de productos, añade "p1", "p2", y "p3" utilizando el método add(), y luego realiza las siguientes acciones: muestra el primer elemento ("p1") sin eliminarlo usando el método peek(), muestra y elimina el primer elemento ("p1") con el método poll(), y finalmente imprime los elementos restantes en la cola, que son "p2" y "p3", utilizando el método forEach().

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
```

Explicación:

El código imprime dos resultados. Primero, muestra la suma de los números 2, 3 y 5, que da 10. Luego, concatena el símbolo "+" con el número 2 y el resultado de multiplicar 3 por 5 (que es 15), produciendo "+215".