Natalia Denise Fuentes Rubio
Semana 4 – Evidencias
Academia Java

1. ¿Qué arroja?

```
public class Main {
public static void main(String[] args) {
                String[] at = {"FINN", "JAKE"};
                for (int x=1; x<4; x++){
                for (String s : at){
                System.out.println(x + " "+ s);
                if(x==1){
                break;
                                }
                        }
                }
        }
}
```

Resultado: 1 FINN 2 FINN 2 JAKE 3 FINN 3 JAKE

Se tiene un arreglo con dos elementos, FINN y JAKE, en el primer bucle se declara su ejecución mientras x tenga los valores de 1, 2 y 3. El segundo bucle iterara los elementos del arreglo. Cuando x = 1, se imprime 1 FINN y llega el break. Cuando x= 2 se imprime 2 FINN y 2 JAKE. Cuando x = 3, imprimirá 3 FINN y 3 JAKE.

2. ¿Que 5 líneas 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;} Sobrecarga del método, por cambio de parámetro

private int lightsaber (long x){return 0;} No se está sobrescribiendo el método, al tener otro parámetro se trata de un método independiente.

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

protected long lightsaber (int x, int y){return 0;} Tiene dos parámetros int, lo que lo hace una sobrecarga del método.

public int lightsaber (int x){return 0;} Sobrescribe el método, y lo cambia de protected a public.

protected long lightsaber (long x){return 0;} Valido 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();
        }
  }
```

Resultado: 15 , 16 , 6

La clase tiene tres atributos públicos para definir características de un ratón. En el primer constructor, de reciben dos parámetros, el peso y cantidad de dientes, el siguiente constructor define los bigotes como 6.
En el constructor final se asignan los valores a peso, dientes y bigotes.
En el print, se imprimen los valores establecidos a lo largo del código; peso 15, dientes 16 y bigotes 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();
        }
    }
```

Resultado: arachnid spider s s

Se va a crear una nueva instancia de Spider al llamar al constructor Arachnid de la clase padre, dicho constructor va a imprimir arachnid, luego se ejecuta el constructor Spider y al ejecutarse imprime spider.
Al llamar al método run el atributo type de la instancia Spider se cambia a s, luego se imprimen los valores de this.type y super.type y como ambos son s, se imprimen

5. Resultado

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

Resultado: 2 2

Se declara una variable b y se inicializa en 4. Con el operador b—se decrementa en 1 su valor, por lo que ahora es 3, posterior a esto, se realiza otro decremento por lo que el valor final de b será 2 y se imprime en ambas impresiones.

6. Resultado

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

Resultado: 997, 997

Se declara la variable ov en 999, primera tendrá un decremento con el operador ov--, lo que convertirá la variable a 998. Posteriormente tiene otro decremento lo que lo convierte en 997 y este valor se imprime. Posteriormente se vuelve a imprimir el valor de ov.

7. 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";
    }
}
```

Resultado: 1, 4, 2

Tenemos el método String overload con cuatro diferentes firmas, cada uno devuelve un valor diferente. En el método main se cuenta con tres diferentes llamadas, lo que el código hará es ver con cual de los métodos existen las coincidencias. Para el primer caso de valor a, se llamará al primer método que regresa 1, para el caso de a y b, se llamará al método cuatro regresando el valor de 4 y para el valor de a, b y c se llamará al segundo método ya que acepta un número variable de Strings, por lo que la tercer salida será un 2.

8. 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();
	}
}
```

Resultado: ClassCastException

Se produce cuando se intenta realizar una conversión de tipos entre clases no relacionadas en una jerarquía de herencia. En este caso, no se puede realizar el cambio de Base1 a Base2, ya que, se quiere convertir obj a Base2, pero obj es en realidad una instancia de Base1 no de Base2.

9. 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);
	}
}
```

Resultado: El código no compila

El error se encuentra en la línea String anotherFish = numFish +1; ya que se intenta asignar una suma aritmética a una variable de tipo String.

10. 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);
	}
}
```

Resultado: 7 7

Number1 es de tipo binario debido al prefijo 0b, el número 0111 equivale al 7.
Number2 es de tipo octal debido al 0 que lleva al inicio, su equivalente es el número 29952.
Al regresar las impresiones, ambos imprimirán, 7 ya que en ambos casos, se llama al number1

11. 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();
	}
}
```

Resultado: 20

En el método calc se multiplica una variable num por 10. En la clase Calculator hay una variable de instancia definida en 100, sin embargo, la variable que entrará en el método calc será el 2 que se actualiza en el main, por lo que se imprimirá 20.

12. ¿Qué aseveraciones son correctas?

```
import java.lang
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. Si se omite el java.util.Random no compilará el código, ya que se debe importar la clase Random.
* java.lang and util.random are redundant. java.util.Random se tiene que importar de forma explícita.
* you dont need to import java.lang. java.lang se importa automáticamente.

13. Resultado

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

Resultado: 10, 12

Primero se imprimirá 10 porque el incremento ocurre después de que se imprima el valor. Para el segundo valor impreso ya está incrementado a 11, y antes de imprimirse se vuelve a incrementar, por lo que el siguiente valor impreso será 12.

14. 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);
        }
}
```

Resultado: sg = 5

Al inicio se inicializan 3 variablles, mn con 11, sg con 0 y ht solo definida como tipo short.

En el bucle for se declara hr = mn con un valor de 11, el bucle se ejecutará en decremento hasta que el valor de hr sea mayor que 6 y sg se incrementará en 1 cada vez.
De esta forma cuando hr sea 6, sg será 5.

15. ¿Cuáles son verdad?

- An ArrayList is mutable. Un ArrayList si es mutable, se pueden agregar, eliminar o modificar sus elementos.
- An Array has a fixed size. Una vez definido, su tamaño será fijo.
- An array is mutable. Si es mutable en cuanto a modificar sus elementos, más no su tamaño.
- An array allows multiple dimensions. Un array puede tener más de una dimensión.
- An arrayList is ordered. Se mantienen los elementos en orden en el que fueron añadidos.
- An array is ordered. Se mantienen en orden los elementos según el orden de almacenamiento.

16. Resultado

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

Errores de compilación, necesita un bool el while

No se puede convertir un int en boolean, se requiere declararlo en el while para realizar la conversión.

17. 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

En filter no se afecta al Stream original ya que lo Streams son inmutables. Se debe asignar el resultado de la operación a una nueva variable o encadenar la operación.

18. 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

Object es la superclase de todas las clases en Java, incluidas las interfaces, esto quiere decir que puede contener cualquier tipo de objeto en Java, incluidos los que son instancias de clases que implementan interfaces.

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

int

Al realizar operaciones entre tipos de datos pequeños como el byte, short o char, el resultado lo da en automático en int, ya que, Java lo promueve para evitar problemas de desbordamiento y pérdida de precisión.

20. The standart application programmmming interface for accesing databases in java?

JDBC

Es una API que permite interactuar con bases de datos relacionales. Proporciona un conjunto de interfaces y clases que permiten conectarse a una base de datos, ejecutar consultas y actualizar datos, y gestionar transacciones.

21. 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.

El uso de paquetes permite organizar el código de manera estructurada y también proporciona un mecanismo para controlar el acceso a clases, métodos y datos. Específicamente el control de acceso como public, protected, private y default. Permiten definir qué partes del código son accesibles desde otras de el mismo o por fuera del paquete.

22. Forma correcta de inicializar un boleano

boolean a = (3>6);

Un bollean puede tomar dos valores true o false. En esta expresión dará un false ya que 3 no es mayor que 6.

23. Resultado

```
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();
	}
}
```

Resultado: RunRimeException

Cuando se ejecute el programa, será lanzada una RunTimeException en doSomething y será impresa en exception. Aquí se muestra cómo manejar excepciones unchecked en Java usando try-catch, y cómo la excepción se maneja de manera controlada.

24. Resultado

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

Resultado: Wont compile

Hay un error en interviewConducted en Manager, al ser un método de una interfaz, el método debe ser public en la clase que implementa 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);
	}
}
```

Resultado: Spider, Arth

Spider hereda el método printName de Arthropod y define su propio método cambiando los argumentos, generando una sobrecarga del método.

Cuando se llama el método printName de Spider con valor de 4, tendrá de salida Spider. Cuando se llame el método printName de Spider con valor de 9.0, tendrá de salida Arth por ser un valor de tipo double.

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]);
		}
	}
}
```

Resultado: Wed, Wed, Wed

Se define una enumeración Days con los valores Mon, Tue, Wed. Dentro del main, se itera sobre los valores de Days, con cada iteración se crea un array d2 que contendrá todos los valores de la enumeración Days. Posterior a esto, imprimirá el valor de la posición 2 de array d2, el cuál es Wed en cada una de sus tres iteraciones.

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);
	}
}
```

Resultado: true 20 false

x es true porque el resultado de true ^ false es true. y sigue siendo 20, ya que no se modifica.
z se convierte en false debido a la asignación (z = 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) {}
}
```

Resultado: 2 4 6 8 5

Hay bloques estáticos, los cuales se ejecutan en orden de aparición, por lo que las primeras salidas serán 2, 4. Posterior a esto, se crea la instancia new InitializacionOrder en el bloque estático y se ejecutan en el orden de aparición que es 6, 8. Por último se ejecuta el constructor que da una salida de 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");
		}
	}
}
```

Resultado: They dont match

En la comparación se evalúa si las referencias message1 y message2 son iguales en la memoria, pero en este caso apuntan a objetos diferentes ya que message1 es de tipo String y message2 es un new String.

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");
	}
}
```

Resultado: 1 2 4 NullPointerException

En el método run se imprime el valor 1, antes de la excepción se imprime el 2, antes del bloque catch se imprime el 4 y posterior a esto, se captura la NullPointerException.

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

Resultado: arroja 1

Como inserta un dato en la base de datos, 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;
	}
}
```

Resultado: the profits are 676000, 18330000, 27980000

MarvelClass en la clase base de MarvelMovieA y MarvelMovieB, cada clase tiene un método getHash. ab1.getHash llama al método getHash de la clase MarvelClass, que devuelve 676000.

ab2.getHash llama al método getHash de la clase MarvelMovieA, que devuelve 18330000.
ab3.getHas llama al método getHash de la clase MarvelMovieB, que 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]);
		}
	}
}
```

Resultado: 4 An arrayindexoutofbondsexception

Se declara un arreglo con 4 elementos. Se crea el bucle for que itera desde 0 hasta que sea menor o igual que la longitud. En la impresión del arreglo, intentará imprimir la posición i, los valores válidos serían 0, 1, 2 y 3. Al intentar iterar en 4 arrojará la excepción ArrayIndexOutOfBoundsException.

34. Pregunta

```
class Menu {
	public static void main(String[] args) {
		String[] breakfast = {"beans", "egg", "ham", "juice"};
		for (String rs : breakfast) {
			int dish = 2;
			while (dish < breakfast.length) {
				System.out.println(rs + "," + dish);
				dish++;
			}
		}
	}
}
```

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

El código tiene una clase Menu con método main que va a iterar a través de un arreglo de cadenas y utiliza un while anidado para imprimir los elementos. Por cada elemento del array breakfast, se imprimirá su valor combinado con cada uno de los valores 2 y 3.

35. Which of the following statement are true:

* string builder es generalmente más rápido qué string buffer. Ambos son muy similares en muchos aspectos, sin embargo, el stringBuilder es más rápido porque no tiene la sobrecarga adicional de sincronización que posee StringBuffer.
* string buffer is threadsafe; stringbuildder is not. StringBuffer es sincrnizado, por lo que es seguro para el uso en múltiples hilos, ya que garantiza que solo un hilo puede acceder al StringBuffer a la vez.

36. Pregunta

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

Resultado: compilation fail

El método equals debería verificar que el argumento es una instancia de CustomKeys antes de hacer la conversión. La comparación ((CustomKeys)o).key == this.key compara las referencias de los objetos Integer en lugar de sus valores. Cuando se sobrescribe equals, también se debe sobrescribir hashCode para mantener la consistencia, de modo que dos objetos que son iguales según equals también tengan el mismo código hash.

37.The catch clause is of the type:

Throwable. Es la superclase de todas las excepciones y errores en Java. Se puede usar en una cláusula catch para capturar tanto excepciones como errores, aunque esto no es recomendable en la mayoría de los casos.
Exception but NOT including RuntimeException. Incluye todas las excepciones verificadas, pero excluye las excepciones no verificadas. Se puede capturar cualquier excepción que sea una subclase de Exception pero no de RuntimeException usando una cláusula catch específica.
CheckedException. Incluye todas las excepciones verificadas.
RunTimeException. . Se puede capturar cualquier excepción que sea una subclase de Exception pero no de RuntimeException usando una cláusula catch específica.

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

Es útil para simplificar la iteración sobre colecciones y arrays en Java, pero no es adecuado para operaciones que modifiquen la colección durante la iteración. Para estas operaciones, un iterador o un bucle for tradicional son opciones más apropiadas.

39. Which of the following methods may appear in class Y, which extends x ?

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

Cuando se extiende una clase en Java, una subclase puede incluir métodos que cumplen con las siguientes reglas en relación con la clase base: Métodos que se heredan, Sobreescritura de métodos, adición de nuevos métodos. La clase Y que extiende la clase X puede incluir:
Métodos heredados y sobrescritos de X.
Nuevos métodos que no están 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");
		}
	}
}
```

Resultado: Equal; respuesta: s1.equalsIgnoreCase(s2)

El programa realiza una comparación entre el valor de s1 y s2. En la sentencia if, se aplica un equalsIgnoreCase, lo que hará que no se distingan entre mayúsculas y minúsculas, por lo que s1 y s2 serán iguales, dando Equal como resultado.

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);
		}
	}
}
```

Resultado: apple, banana, grapes, pears

Se declara un array de fruits con 4 elementos. Con Arrays.sort(fruits, (a, b) -> a.compareTo(b)); ordena el arreglo fruits en orden alfabético utilizando compareTo, que compara las cadenas lexicográficamente.
La expresión (a, b) -> a.compareTo(b) es una función lambda que define la forma en que se comparan dos elementos del arreglo.
Se recorre el arreglo ordenado con un bucle for-each y se imprimen los elementos en ordenados alfabéticamente.

42. Pregunta

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

Resultado: this code will trow an ArrayIndexOutOfBoundsExpression

Al ejecutar el código será lanzada la excepción ArrayIndexOutOfBoundsException porque se intenta acceder a un índice que no existe en el arreglo (-1). En Java, los índices de los arreglos deben estar dentro de los límites del arreglo, que en este caso son 0, 1, y 2.

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);
	}
}
```

Resultado: 0

En la clase Salmon, se inicializa una instancia de tipo int que en automático es un valor 0, en caso de no ser definido. El constructor de Salmon es en realidad un método, por lo que no tendrá valor de retorno (un constructor no debe especificar un tipo de retorno).
Salmon s = new Salmon(); crea una nueva instancia de la clase Salmon, pero dado que el método Salmon no es un constructor, el campo count no se inicializa a 4, sino que, permanecera con el valor predeterminado 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;
}
```

Resultado: Hay que corregir linea 6; c1 se le asigna c2 pero c2 aún no se declara

En la línea int c1 = c2;, la variable c2 se usa antes de haber sido declarada, lo que provoca un error de compilación.

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);
	}
}
```

Resultado: 611

Se definen tres variables numéricas de diferentes tipos. En la salida, a se incrementa en 1 por lo que será 11, b se mantiene como 20 durante la multiplicación y posterior a esto, se incrementa convirtiéndose en 21, por último, c se mantiene como 30.
Se efectúa primero la multiplicación entre 20*30 dando como resultado 600 y posterior se le suma el valor de a que es 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]+"");
	}
}
```

Resultado: Compilation fails

No se compila debido a un error de sintaxis en el método go. El problema es que el método go tiene un parámetro de tipo varargs (String... y) seguido de un parámetro normal (int x), lo cual no es permitido en Java. Los parámetros de tipo varargs deben ser siempre el último parámetro en la lista de parámetros del método.

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();
	}
}
```

Resultado: plant, leaves, plant, leaves

Tenemos tres clases, Plant que es la superclase, tiene un constructor sin parámetros e imprime plant. La clase Tree hereda de Plant, acepta un String (type) como parámetro y lo imprime. La clase Forest que hereda de Tree, tiene un constructor sin parámetros, sin embargo, llama al constructor de Tree con el argumento leaves.
Al crear una instancia de Forest, primero se llama al constructor de Plant debido a la herencia, dicho constructor va a imprimir “plant”. Luego, el constructor de Tree se ejecuta debido a la llamada a super("leaves");, lo que imprime "leaves". Luego, el new Tree("leaves"); dentro del constructor de Forest crea una nueva instancia de Tree, lo que nuevamente llama al constructor de Plant (imprimiendo "plant") y después imprime "leaves".

48. Pregunta

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

Resultado: true

Se crea una cadena s1 con el valor "hello". s2 se crea como un nuevo objeto String con el valor "hello". El método intern() verifica si el contenido de s2 ya existe en el pool de strings.
Si existe, devuelve una referencia a la cadena existente en el pool. El operador == compara si s1 y s2 apuntan al mismo objeto en la memoria. Dado que s2.intern() hizo que s2 apunte al mismo objeto en el pool de strings que s1, la comparación s1 == s2 será true

49. ¿Cuál de las siguientes construcciones es un ciclo infinito while?:

- while(true);. Es infinito porque la condición true siempre es verdadera. El ciclo se ejecutará indefinidamente. El ; al final del while(true); significa que el cuerpo del ciclo es una instrucción vacía. Esto hace que el ciclo no haga nada, pero sigue siendo un ciclo infinito.
- while(1==1){}. Es infinito porque la condición 1==1 siempre es verdadera. El ciclo se ejecutará indefinidamente. El cuerpo del ciclo {} está vacío, lo que significa que no hace nada, pero sigue siendo un ciclo infinito.

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);
	}
}
```

Resultado: 0

Se inicializan las variables a, b, z, w con sus respectivos valores. El primer bloque if compara si a es igual a b. En este caso, 10 no es igual a 37, por lo que este bloque no se ejecuta y z no se asigna a 3. El bloque else if compara si a es mayor que b. Como 10 no es mayor que 37, este bloque tampoco se ejecuta y z no se asigna a 6. Dado que ninguna de las condiciones es verdadera, z se mantiene en 0. w = 10 * z;. Como z es 0, entonces w será 0 * 10, lo que da como resultado 0. Como z no ha sido modificado, al imprimirse será 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";
	}
}
```

Resultado: Exception StackOverflowError

Provoca una recursión infinita cuando se intenta crear un objeto de la clase course. La clase course tiene una variable de instancia name de tipo String. En el constructor de course, se crea un nuevo objeto de la clase course. Cada vez que se crea un objeto de course, el constructor intenta crear otro objeto de course, lo que provoca un bucle recursivo infinito.

52. Pregunta

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

Resultado: builder fails

La variable a es declarada pero no se inicializa. La línea de impresión intenta llamar al método toString en la variable a, como a no está inicializada provocará un NullPointerException ya que su valor es null, sin embargo, nunca arrojará esta excepción ya que el código no podrá compilarse.

53. Pregunta

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

Resultado: 10 + 235

En la primera impresión, si se realizará la operación sumatoria dando como resultado un 10. En la segunda expresión se concatenarán los números debido a la presencia del "+", por lo que saldrá +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");
	}
}
```

Resultado: Here1 , Here 3

En el main se inicializan las variables a y b con valor de 2 cada una, posteriormente se inicia una serie de condiciones, en la primera condición se evalúa que a y b sean iguales, en este caso si lo son así que se imprimirá Here1. En la segunda se evalúa si a y b son diferentes, al no ser cierto, no se realizará la impresión. En la tercera condición, se evalúa si a es mayor o igual que b, al ser ambas iguales, se imprimirá Here3.

55. Pregunta

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

Resultado: builder fails

La clase Main está intentando llamar al método count(int x, int y) desde el método main, pero el método count no es estático. El método count debería ser accesible desde la clase Main.

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");
	}
}
```

Resultado: magic town

Se generan varias sobrecargas al método main, sin embargo, public static void main (String [] args), es el que se ejecutará al correr el código , ya que es el único que coincide con el método entrada estándar de Java, por lo que va a regresar 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);
	}
}
```

Resultado: 2, 1

Se inicializa a en 0, en la primera impresión se le efectúa un post – decremento, al ser post, en el momento de la operación a seguirá siendo 0, por lo que la primera salida es 2. En la segunda impresión, a ya ha sufrido el post – decremento, por lo que su valor ahora es de 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);
	}
}
```

Resultado: builder fails

La declaración List<E> usa un tipo genérico E que no está definido en ninguna parte del código. Esto genera un error de compilación porque no se sabe qué tipo de datos se deberían almacenar en la lista.

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);
	}
}
```

Resultado: break es una palabra reservada

break es una palabra reservada utilizada para salir de bucles y switch statements. No puede ser usada como nombre de un método, lo que genera un error de compilación.

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);
	}
}
```

Resultado: 3, 4

En new App(100) el 100 es un entero de tipo int, el constructor que acepta int es el que devuelve un 3 de valor, por lo que éste será el primer valor impreso. En el new App(100L) 100L es un entero de tipo long, el constructor que lo acepta es el App(Object num), dado que long puede ser convertido a Object.

61. Pregunta

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

Resultado: everething

Se declará la variable i en 42, luego se implementa el operador ternario para asignar un valor de tipo string a s. Si i es menor que 40, se asigna la cadena "life". Si i es mayor que 50, se asigna la cadena "universe". Si ninguna de las condiciones anteriores es verdadera (es decir, si i está entre 40 y 50 inclusive), se asigna la cadena "everything". Finalmente, se imprime el valor de s, que al ser 42 será 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));
```

```
                }
        }
}
```

Resultado: java: unreported exception java.text. ParseException; must be caught or declared to be thrown

NumberFormat no se puede resolver, por lo que el programa lanzará un error.

63. Pregunta

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

Resultado: true

Se declarán las variables s1 y s2, a s1 se le asigna el valor de OCAJP, a s2 se le concatena el valor de OCAJP y toma este valor. Al momento de realizar la comparación, el resultado será true ya que es verdadera.

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;
        }
}
```

Resultado: Error de compilacion - java: reached end of file while parsing

Switch no acepta enum como tipo de dato, por lo que las líneas case score < 70: y case score >= 70: marcarán un error.

65. Pregunta

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

Resultado: -100

Se declara una variable a con un valor de 100. Posteriormente se le realiza una operación, primero se va a cambiar el signo a -. Antes de hacer el incremento, se va a imprimir el -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");
		}
	}
}
```

Resultado: Error de compilacion - java: incompatible types: posible lossy conversion from int to byte

Se declara una variable var de tipo byte con valor 100. El bloque switch toma la variable var y evalúa su valor en función de los case. Si var es igual a 100, se ejecuta el código dentro del case 100. Si var es igual a 200, se ejecuta el código dentro del case 200. Si el valor de var no coincide con ninguno de los case, se ejecuta el bloque default. Una variable de tipo byte tiene un rango de valores de -128 a 127. El valor 200 está fuera del rango permitido para un tipo byte. Este valor es demasiado grande para ser representado por un byte, por lo que la línea case 200: causará un error de compilación.

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");
	}
}
```

Resultado: ClassCastException

La clase B extiende a la clase A. La clase B hereda el método print() de A, pero lo sobrescribeEn el main, se crea una instancia de A llamada obj1. Luego, se intenta hacer un downcast de obj1 a un objeto de tipo B y asignarlo a obj2. Se llama al método print() a través de obj2. El obj1 es una instancia de A, sin embargo, al hacer el downcasting de onj1 de B, resultará en error ya que es una instancia de A.

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;
		}
	}
}
```

Resultado: ANY FRUIT WILL DO, APPLE, MANGO, BANANA

Se declara un String fruit con el valor de mango. En el switch se compara el valor de fruit con cada uno de los casos específicados, ya que o hay breaks hasta después de BANANA, el código correra hasta el final. La variable es "mango", pero en el switch no hay coincidencia exacta con "Mango" debido a la diferencia en mayúsculas/minúsculas. Dado que no hay coincidencias se ejecutará el bloque default primero y al no haber un break el control del flujo caerá a través de los siguientes casos.

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());
        }
}
```

Resultado: compilation fails

La clase Animal tiene un campo name que se inicializa a través del constructor. El método getName() devuelve el valor del campo name. Dog extiende Animal y tiene un campo adicional breed. Tiene dos constructores, en el primero breed, no llama al constructor de la superclase. Un constructor de subclase siempre debe llamar explícitamente a un constructor de la superclase, ya sea directamente o indirectamente.

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));
		}
	}
}
```

Resultado: 333.6789, 234.111

Se crea un array de String que contiene dos valores numéricos con más de dos decimales. NumberFormat.getInstance() crea una instancia del formateador de números. nf.setMaximumFractionDigits(2) configura el formateador para mostrar hasta dos dígitos decimales. El bucle recorre cada cadena en el array sa, la analiza utilizando nf.parse(s), el cual convierte la cadena en un Number. Dado que el código usa valores "333.6789" y "234.111" con más de dos dígitos decimales, el método parse() los convertirá en Double.

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);
		});
	}
}
```

Resultado: p1, p1, , p2, p3.

Se crea una instancia de ArrayDeque y se asigna a una variable de tipo Queue. ArrayDeque es una implementación de cola que proporciona operaciones de cola y pila eficientes. Se añaden tres elementos a la cola: "p1", "p2", y "p3". Estos elementos se almacenan en el orden en que se agregan. products.peek() devuelve el primer elemento sin eliminarlo; en este caso, devolverá "p1", products.poll() devuelve y elimina el primer elemento de la cola. Por lo tanto, eliminará "p1" y devolverá "p1". Se imprime una línea en blanco para separar los resultados. products.forEach() itera sobre los elementos restantes en la cola y los imprime. Después de la llamada a poll(), la cola contiene "p2" y "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);
	}
}
```

Resultado: 10 + 215

En la primera impresión, regresará el valor de la suma de 2, 3 y 5 lo que dará 10 como resultado. En la segunda impresión, se realizará una concatenación; se efectuará la multiplicación de 3*5, por lo que se concatenará 215.