

Nombre: Juan Ivan Velazquez Cabello

Semana: Tarea semana 5

Documento: ejercicios-propuestos.docx.....	2
Examen Java V3.....	23
Explicación:.....	32
Errores.pdf.....	33
Preguntas que vinieron.docx.....	39
Diferencia entre parse y format:.....	45
Preguntas que vinieron APX.docx.....	55
Tipos de datos permitidos en un switch en Java:.....	55
Ayuda con el contrato de equals:.....	60
Reglas para métodos que no retornan ningún valor:.....	62

Documento: ejercicios-propuestos.docx

Ejercicio 1.

Considera el siguiente bloque de código:

```
class Animal {  
  
    void makeSound() throws Exception {  
  
        System.out.println("Animal makes a sound");  
  
    }  
  
}  
  
class Dog extends Animal {  
  
    void makeSound() throws RuntimeException {  
  
        System.out.println("Dog barks");  
  
    }  
  
}  
  
public class Main {  
  
    public static void main(String[] args) {  
  
        Animal myDog = new Dog();  
  
        try {  
  
            myDog.makeSound();  
  
        } catch (Exception e) {  
  
            System.out.println("Exception caught");  
  
        }  
    }  
}
```

```
 }  
 }
```

¿Cuál sería la salida en consola al ejecutar este código?

1- Dog barks

- 2- Animal makes a sound
- 3- Exception caught
- 4- Compilation error

Ejercicio 2

Ejercicio de Hilos (Threads)

Considera el siguiente bloque de código:

```
class Animal {  
  
    void makeSound() throws Exception {  
  
        System.out.println("Animal makes a sound");  
  
    }  
  
}  
  
  
class Dog extends Animal {  
  
    void makeSound() throws RuntimeException {  
  
        System.out.println("Dog barks");  
  
    }  
  
}
```

```
class MyThread extends Thread {  
  
    public void run() {  
  
        System.out.println("Thread is running");  
  
    }  
  
}  
  
  
  
public class Main {  
  
    public static void main(String[] args) {  
  
        Thread t1 = new MyThread();  
  
        Thread t2 = new MyThread();  
  
        t1.start();  
  
        t2.start();  
  
    }  
  
}
```

¿Cuál sería la salida en consola al ejecutar este código?

- 1- Thread is running (impreso una vez)
- 2- Thread is running (impreso dos veces)
- 3- Thread is running (impreso dos veces, en orden aleatorio)**
- 4- Compilation error

Ejercicio 3

Ejercicio de Listas y Excepciones

Considera el siguiente bloque de código:

```
public class Main {  
  
    public static void main(String[] args) {  
  
        List<Integer> numbers = new ArrayList<>();  
  
        numbers.add(1);  
  
        numbers.add(2);  
  
        numbers.add(3);  
  
        try {  
  
            for (int i = 0; i <= numbers.size(); i++) {  
  
                System.out.println(numbers.get(i));  
  
            }  
  
        } catch (IndexOutOfBoundsException e) {  
  
            System.out.println("Exception caught");  
  
        }  
  
    }  
}
```

¿Cuál sería la salida en consola al ejecutar este código?

1- 1 2 3 Exception caught

2- 1 2 3

3- Exception caught

4- 1 2 3 4

ejercicio 4

Ejercicio de Herencia, Clases Abstractas e Interfaces

Considera el siguiente bloque de código:

```
import java.util.ArrayList;
import java.util.List;

interface Movable {
    void move();
}

abstract class Vehicle {
    abstract void fuel();
}

class Car extends Vehicle implements Movable {
    void fuel() {
        System.out.println("Car is refueled");
    }

    public void move() {
        System.out.println("Car is moving");
    }
}

public class Main {
    public static void main(String[] args) {
        Vehicle myCar = new Car();
    }
}
```

```
    myCar.fuel();

    (Movable) myCar).move();

}

}
```

¿Cuál sería la salida en consola al ejecutar este código?

1- Car is refueled Car is moving

2- Car is refueled

3- Compilation error

Ejercicio 5

Ejercicio de Polimorfismo y Sobrecarga de Métodos

Considera el siguiente bloque de código:

¿Cuál sería la salida en consola al ejecutar este código?

```
class Parent {

    void display(int num) {
        System.out.println("Parent: " + num);
    }

    void display(String msg) {
        System.out.println("Parent: " + msg);
    }
}
```

```
class Child extends Parent {

    void display(int num) {
        System.out.println("Child: " + num);
    }
}
```

```
    }

}

public class Main {

    public static void main(String[] args) {

        Parent obj = new Child();

        obj.display(5);

        obj.display("Hello");

    }
}
```

¿Cuál sería la salida en consola al ejecutar este código?

1- Child: 5 Parent: Hello

2- Parent: 5 Parent: Hello

3- Child: 5 Child: Hello

4- Compilation error

Ejercicio de Hilos y Sincronización

Considera el siguiente bloque de código:

```
class Counter {  
  
    private int count = 0;  
  
    public synchronized void increment() {  
  
        count++;  
    }  
  
    public int getCount() {  
  
        return count;  
    }  
}  
  
class MyThread extends Thread {  
  
    private Counter counter;  
  
    public MyThread(Counter counter) {  
  
        this.counter = counter;  
    }  
  
    public void run() {  
  
        for (int i = 0; i < 1000; i++) {  
  
            counter.increment();  
        }  
    }  
}
```

```
public class Main {  
  
    public static void main(String[] args) throws InterruptedException  
{  
  
    Counter counter = new Counter();  
  
    Thread t1 = new MyThread(counter);  
  
    Thread t2 = new MyThread(counter);  
  
    t1.start();  
  
    t2.start();  
  
    t1.join();  
  
    t2.join();  
  
    System.out.println(counter.getCount());  
  
}  
  
}
```

¿Cuál sería la salida en consola al ejecutar este código?

1- 2000

2- 1000

3- Variable count is not synchronized

4- Compilation error

Ejercicio 7

Ejercicio de Listas y Polimorfismo

Considera el siguiente bloque de código:

```
import java.util.ArrayList;  
  
import java.util.List;  
  
  
class Animal {  
  
    void makeSound() {  
  
        System.out.println("Animal sound");  
  
    }  
  
}  
  
  
class Dog extends Animal {  
  
    void makeSound() {  
  
        System.out.println("Bark");  
  
    }  
  
}  
  
  
class Cat extends Animal {  
  
    void makeSound() {  
  
        System.out.println("Meow");  
  
    }  
  
}
```

```
public class Main {  
  
    public static void main(String[] args) {  
  
        List<Animal> animals = new ArrayList<>();  
  
        animals.add(new Dog());  
  
        animals.add(new Cat());  
  
        animals.add(new Animal());  
  
        for (Animal animal : animals) {  
  
            animal.makeSound();  
  
        }  
  
    }  
}
```

¿Cuál sería la salida en consola al ejecutar este código?

1- Animal sound Animal sound Animal sound

2- Bark Meow Animal sound

3- Animal sound Meow Bark

4- Compilation error

Ejercicio 8

Ejercicio de Manejo de Excepciones y Herencia

Considera el siguiente bloque de código:

```
import java.io.FileNotFoundException;
import java.io.IOException;

class Base {
    void show() throws IOException {
        System.out.println("Base show");
    }
}

class Derived extends Base {
    void show() throws FileNotFoundException {
        System.out.println("Derived show");
    }
}

public class Main {
    public static void main(String[] args) {
        Base obj = new Derived();
        try {
            obj.show();
        } catch (IOException e) {
            System.out.println("Exception caught");
        }
    }
}
```

```
    }  
}  
}
```

¿Cuál sería la salida en consola al ejecutar este código?

1- Base show

2- Derived show

3- Exception caught

4- Compilation error

Ejercicio 9

Ejercicio de Concurrencia y Sincronización

Considera el siguiente bloque de código:

```
class SharedResource {  
  
    private int count = 0;  
  
    public synchronized void increment() {  
  
        count++;  
    }  
  
    public synchronized void decrement() {  
  
        count--;  
    }  
  
    public int getCount() {  
  
        return count;  
    }  
}  
  
class IncrementThread extends Thread {
```

```
private SharedResource resource;

public IncrementThread(SharedResource resource) {

    this.resource = resource;
}

public void run() {

    for (int i = 0; i < 1000; i++) {

        resource.increment();
    }
}

}

class DecrementThread extends Thread {

private SharedResource resource;

public DecrementThread(SharedResource resource) {

    this.resource = resource;
}

public void run() {

    for (int i = 0; i < 1000; i++) {

        resource.decrement();
    }
}

}

public class Main {

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

```
SharedResource resource = new SharedResource();

Thread t1 = new IncrementThread(resource);

Thread t2 = new DecrementThread(resource);

t1.start();

t2.start();

t1.join();

t2.join();

System.out.println(resource.getCount());

}

}
```

¿Cuál sería la salida en consola al ejecutar este código?

1- 1000

2- 0

3- -1000

4- Compilation error

Ejercicio 10

Ejercicio de Generics y Excepciones

Considera el siguiente bloque de código:

```
class Box<T> {

    private T item;

    public void setItem(T item) {

        this.item = item;
    }

    public T getItem() throws ClassCastException {

        if (item instanceof String) {

            return (T) item; // Unsafe cast
        }

        throw new ClassCastException("Item is not a String");
    }
}

public class Main {

    public static void main(String[] args) {

        Box<String> stringBox = new Box<>();

        stringBox.setItem("Hello");

        try {

            String item = stringBox.getItem();

            System.out.println(item);

        } catch (ClassCastException e) {
    }
}
```

```
        System.out.println("Exception caught");

    }

}

}
```

¿Cuál sería la salida en consola al ejecutar este código?

1- Hello

2- Exception caught

3- Compilation error

4- ClassCastException

Ejercicio 11

Considera el siguiente bloque de código:

```
public class Main {  
  
    public static void main(String[] args) {  
  
        Padre objetoPadre = new Padre();  
  
        Hija objetoHija = new Hija();  
  
        Padre objetoHija2 = (Padre) new Hija();  
  
        objetoPadre.llamarClase();  
  
        objetoHija.llamarClase();  
  
        objetoHija2.llamarClase();  
  
        Hija objetoHija3 = (Hija) new Padre();  
  
        objetoHija3.llamarClase();  
  
    }  
  
}  
  
  
class Hija extends Padre {  
  
    public Hija() {  
  
    }  
  
    @Override  
  
    public void llamarClase() {  
  
        System.out.println("Llame a la clase Hija");  
  
    }  
  
}
```

```
class Padre {  
    public Padre() {  
    }  
    public void llamarClase() {  
        System.out.println("Llame a la clase Padre");  
    }  
}
```

a)

Llame a la clase Padre

Llame a la clase Hija

Llame a la clase Hija

Error: java.lang.ClassCastException

b)

Llame a la clase Padre

Llame a la clase Hija

Llame a la clase Hija

Llame a la clase Hija

c)

Llame a la clase Padre

Llame a la clase Hija

Llame a la clase Hija

Llame a la clase Padre

Ejercicio 12

Considera el siguiente bloque de código:

```
public class Main {
    public static void main(String[] args) {
        Animal uno = new Animal();
        Animal dos = new Dog();
        uno.makeSound();
        dos.makeSound();
        Dog tres = (Dog) new Animal();
        tres.makeSound();
    }
}

class Animal {
    void makeSound() {
        System.out.println("Animal sound");
    }
}

class Dog extends Animal {
    void makeSound() {
        System.out.println("Wau Wau");
    }
}
```

- 1) Animal sound Wau Wau compilation error
- 2) Compilation Error java.lang.ClassCastException**
- 3) Animal sound Wau Wau Animal sound
- 4) Animal sound

Ejercicio 13

Considera el siguiente bloque de código:

```
public class Main {

    public static void main(String[] args) {
        Cambios uno = new Cambios();
        int x = 1;
        String hola = "hola";
        StringBuilder hola2 = new StringBuilder("hola2");
        Integer x2 = 4;
        uno.makeSound(x, hola);
        uno.makeSound(x2, hola2);
        System.out.println("Cambios?: " + x + ", " + hola + ", " + x2 +
", " + hola2);
    }

}

class Cambios {

    void makeSound(int x, String s) {
        s = "cambiando string";
        x = 5;
    }

    void makeSound(Integer x, StringBuilder s) {
        x = 9;
        s = s.delete(0, s.length());
    }
}
```

- 1) Compilation error
- 2) Cambios?: 1,hola,4,
- 3) Cambios?: 1,hola,4,hola2
- 4) Cambios?: 5,cambiando string,9,

Ejercicio 14

Considera el siguiente bloque de código:

```
interface i1 {  
    public void m1();  
}  
  
interface i2 extends i1 {  
    public void m2();  
}  
  
class animal implements i1, i2 {  
    //¿Qué métodos debería implementar la clase animal en este  
    //espacio?  
}
```

- 1) solo m1
- 2) m1 y m2**
- 3) ninguno
- 4) error compilación

Examen Java V3

Ejercicio 1

The feature which allows different methods to have the same name and arguments type, but the different implementation is called?

Pick one of the choices:

- Overloading
- **Overriding**
- Java does not permit methods with same and type signature
- None of the above

Ejercicio 2

What is the following for loop output?

```
public class Main {  
  
    public static void main(String[] args) {  
  
        for(int i=10, j=1; i>j; i--, j++) {  
  
            System.out.println(j % i);  
  
        }  
  
    }  
}
```

Pick one of the choices:

- 12321
- **12345**
- 11111
- 00000

Ejercicio 3

We perform the following sequence of actions:

1. Insert the following elements into a set: 1,2,9,1,2,3,1,4,1,5,7.
2. Convert the set into a list and sort it in ascending order.

Which option denotes the sorted list?

Pick one of the choices:

- {1, 2, 3, 4, 5, 7, 9}
- {9, 7, 5, 4, 3, 2, 1}
- {1, 1, 1, 1, 2, 2, 3, 4, 5, 7, 9}
- None of the above

Ejercicio 4

What is the output for the below Java code

```
1 public class Main {
2     public static void main(String[] args) {
3         int i = 010;
4         int j = 07;
5         System.out.println(i);
6         System.out.println(j);
7     }
8 }
```

Pick one of the choices:

- 8 7
- 10 7
- Compilation fails with an error at line 3

Ejercicio 5

A public data member with the same name is provided in both base as well as derived classes. Which of the following is true?

Pick one of the choices:

- It is a compiler error to provide a field with the same name in both base and derived class
- The program will compile and this feature is called overloading
- The program will compile and this feature is called overriding
- **The program will compile and this feature is called as hiding or shadowing**

Ejercicio 6

Given three classes A, B and C.

B is a subclass of A

C is a subclass of B

Which one of these boolean expressions is true only when an object denoted by reference o has actually been instantiated from class B, as opposed to from A or C?

Pick one of the choices:

- (o instanceof B) && (! (o instanceof A))
- **(o instanceof B) && (! (o instanceof C))**
- (o instanceof B)
- None of the above

Ejercicio 7

Which statement is true?

Pick one of the choices:

- Non-static member classes must have either default or public accessibility
- All nested classes can declare static member classes
- Methods in all nested classes can be declared static
- **Static member classes can contain non-static methods**

Ejercicio 8

A constructor is called whenever

Pick one of the choices:

- **An object is instanced**
- An object is used
- A class is declared
- A class is used

Ejercicio 9

Which of the following data types in Java are primitive?

Pick one of the choices:

- String
- Struct
- **Boolean**
- **char**

Ejercicio 10

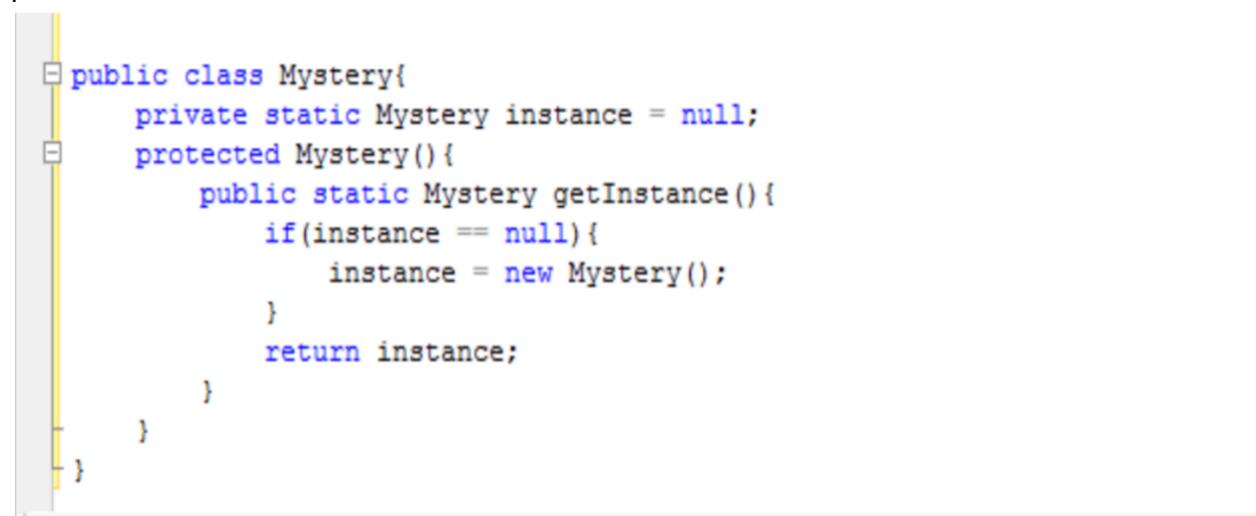
Which of the following are true for Java Classes?

Pick one of the choices:

- The Void class extends the Class class
- The Float class extends the Double class
- The System class extends the Runtime class
- **The Integer class extends the Number class**

Ejercicio 11

The following code snippet is a demonstration of a particular design pattern. Which design pattern is it?



```
public class Mystery{
    private static Mystery instance = null;
    protected Mystery(){
        public static Mystery getInstance(){
            if(instance == null){
                instance = new Mystery();
            }
            return instance;
        }
    }
}
```

A screenshot of a Java code editor showing a code snippet. The code defines a class named 'Mystery' with a private static variable 'instance' initialized to null. It contains a protected constructor and a public static method 'getInstance()' that returns the single instance of the class. The code is syntax-highlighted with blue for keywords like 'public', 'private', 'static', 'protected', 'if', and 'new'. Braces and identifiers are in black.

Pick one of the choices:

- Factory Design Pattern
- Strategy Pattern
- **Singleton**
- Facade Design Pattern

Ejercicio 12

Which of the following Java declarations of the String array are correct?

Pick one of the choices:

- String temp [] = new String {"j", "a", "z"};
- String temp [] = {"j" "b" "c"};
- String temp = {"a", "b", "c"};
- **String temp [] = {"a", "b", "c"};**

Ejercicio 13

Which is true of the following program?

```
public class TestFirstApp {  
    static void doIt(int x, int y, int m) {  
        if(x==5) m=y;  
        else m=x;  
    }  
    public static void main(String[] args) {  
        int i=6, j=4, k=9;  
        TestFirstApp.doIt(i, j, k);  
        System.out.println(k);  
    }  
}
```

Pick one of the choices

- Doesn't matter what the values of i and j are, the output will always be 5.
- Doesn't matter what the values of k and j are, the output will always be 5.
- **Doesn't matter what the values of i and j are, the output will always be 9.**
- Doesn't matter what the values of k and j are, the output will always be 9.

Ejercicio 14

Which of the following statements are correct. Select the correct answer.

- Each Java file must have exactly one package statement to specify where the class is stored.
- If a java file has both import and package statements, the import statement must come before package statement.
- A java file has at least one class defined
- If a java file has a package statement, it must be the first statement (except comments).

Ejercicio 15

What is the output for the following program:

```
public class Constructor {  
    static String str;  
    public void Constructor(){  
        System.out.println("In constructor");  
        Str = "Hello World";  
    }  
    public static void main(String[] agrs){  
        Constructor c = new Constructor();  
        System.out.println(str);  
    }  
}
```

Pick one of the choices

- In Constructor
- Null
- Compilation Fails
- None of the above

Ejercicio 16

Given the following code, what is the most likely result?

```
public class Constructor {  
  
    public static void main(String... args){  
  
        String [] cities = {"hola", "dos"};  
  
    }  
  
    static class MySort implements Comparator{  
  
        public int compare(String a, String b){  
  
            return b.compareTo(a);  
  
        }  
  
    }  
  
}
```

Pick one of the choices

- -1
- 1
- 2
- **Compilation fails**

Ejercicio 17

To delete all pairs of keys and values in a given **HashMap**, which of the following methods should be used? Pick one of the choices

- a) `clearAll()`
- b) `empty()`
- c) `remove()`

d) clear()

Ejercicio 18

Which pattern do you see in the code below: `java.util.Calendar.getInstance()`; Pick one of the choices

a) Singleton Pattern

b) Factory Pattern

c) Facade Pattern

d) Adaptor Pattern

Explicación:

1. Factory Pattern (Patrón de Fábrica):

- Este patrón se utiliza para crear instancias de clases sin exponer al cliente los detalles de implementación.
- En el caso de `Calendar.getInstance()`, el método decide cuál implementación concreta de `Calendar` devolver según la configuración regional del sistema o el tipo de calendario predeterminado (por ejemplo, `GregorianCalendar`).
- El cliente no necesita conocer la implementación específica que está utilizando; simplemente llama al método estático `getInstance()`.

Ejercicio 19

What is the output of the following program:

```
interface BaseI {
    void method();
}

class BaseC {
    public void method() {
        System.out.println("Inside BaseC: :method");
    }
}

public class Prueba extends BaseC implements BaseI {
    public static void main(String[] s) {
        (new Prueba()).method();
    }
}
```

Pick one of the choices

- a) Null
- b) Compilation fails
- c) Inside BaseC: :method
- d) None of the above

Errores.pdf

Ejercicio 1

```
public class Simple {  
  
    public float price;  
  
    public static void main(String[] args) {  
  
        Simple price = new Simple();  
  
        price = 4;  
  
    }  
  
}
```

- Change line 7 to the following: float price = new Simple ();
- Change line 3 to the following: public int price; ✓
- Change line 7 to the following: price = 4f;
- Change line 7 to the following: price.price = 4; ✗
- Change line 7 to the following: int price = new Simple();

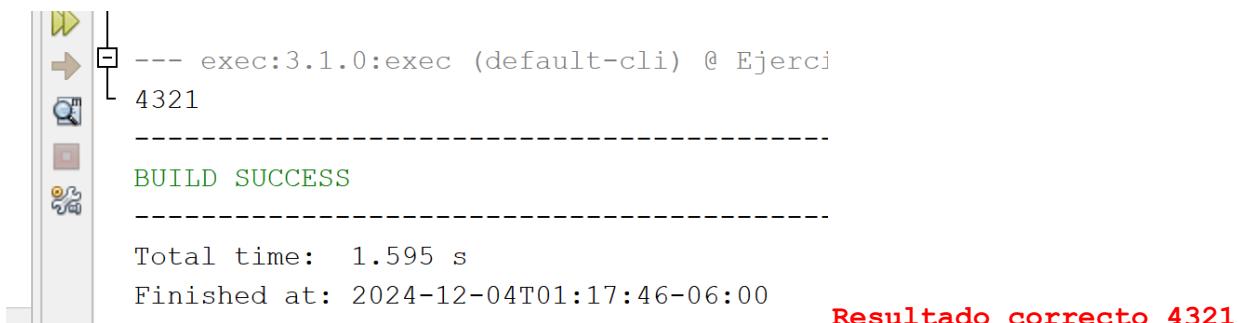
La respuesta correcta es la que está en rojo porque así se accede al atributo price de la clase

Ejercicio 2

What is the result

```
class Person {
    String name = "No name";
    public Person(String nm) {
        name = nm;
    }
}
class Employee extends Person {
    String empID = "0000";
    public Employee(String id) {
        super(id);
        empID = id;
    }
}
public class EmployeeTest {
    public static void main(String[] args) {
        Employee e = new Employee("4321");
        System.out.println(e.empID);
    }
}
```

- Compilation fails because of an error in line 18. ✓
- 0000
- An exception is thrown at runtime.
- 4321 ✗



Ejercicio 3

What is the result

```
public class Lol {
    public static void main(String[] args) {
        try{
            doSomething();
        } catch(RuntimeException e){
            System.out.println(e);
        }
    }
    static void doSomething() {
        if(Math.random()>0.5){
            throw new IOException();
        }
        throw new RuntimeException();
    }
}
```

- Adding throws IOException to the doSoomething() method signature.
- Adding throws IOException to the main() method signature and to the dosomething() method ✗**
- Adding throws IOException to the main() methodIOException ✓
- Adding throws IOException to the main() method signature
- Adding throws IOException to the dosomething() method signature and changing the catch

El resultado correcto es adding throws IOException to the main() method signature and to the dosomething() method

Ejercicio 4

What is the result

22. Which two may precede the word "class" in a class declaration?

- Static ✓
- Local
- Volatile
- Synchronized
- Public ✓

Solo es Public Static no puede ser porque una clase no puede tener el modificador de acceso static.

Ejercicio 5

What is the result

24. Which code fragment is illegal?

- Class Base1 { abstract class Abs1 { } }
- class Base2 { abstract class Abs3 extends Base2 { } }
- class Base3 { abstract int var1 = 89; } ✓
- Abstract class Abs2 { void doit() { } }

Las respuestas correctas si es la que está marcada en azul pero también es la última porque abstract class debe ir con minúsculas además también la primera porque Class esta en mayuscula, si estuvieran escritas con minúsculas solo abriría una incorrecta que es la seleccionada ya que no se puede declarar una variable abstracta.

no es posible declarar una variable como **abstract**. Esto se debe a que el modificador **abstract** está diseñado para ser usado exclusivamente con **clases y métodos**, y tiene un propósito específico: indicar que algo está incompleto y debe ser implementado por una subclase o una clase concreta.

Clases abstractas: Pueden contener métodos abstractos (sin implementación) y métodos concretos (con implementación).

java

Copiar código

```
abstract class Shape {  
    abstract void draw(); // Método abstracto, debe ser implementado  
    por una subclase  
    void rotate() {  
        System.out.println("Rotating the shape");  
    }  
}
```

Métodos abstractos: Sólo pueden declararse dentro de una clase abstracta y deben ser implementados en una subclase concreta.

java

Copiar código

```
abstract class Animal {  
    abstract void makeSound(); // Obligatorio implementar en una  
    subclase  
}
```

Preguntas que vinieron.docx

Ejercicio 1

¿Cuántos errores tiene el siguiente código?

```
1 public class MyClass {  
2  
3     static final String s1 = "Hola";  
4     static final String s2 = "Mundo!";  
5     static final String s3 = "";  
6  
7     static {  
8         s1 = "Hello";  
9     }  
10  
11    static {  
12        s2 = "World";  
13    }  
14  
15    public static void main(String args[]) {  
16  
17        s3 = "Despues";  
18    }  
19}  
20}  
21}
```

respuestas posibles:

0

1

3

Ejercicio 2

¿Qué pasa con el siguiente código?

```
1 public class MyClass {  
2     public static void main(String args[]) {  
3  
4         String Hola = "Hola mundo";  
5  
6         System.out.println(Hola.charAt(11));  
7     }  
8 }  
9
```

Exception in thread "main" java.lang.StringIndexOutOfBoundsException: Index 4 out of bounds for length 10

Ejercicio 3

¿Qué pasa con el siguiente código?

```
1 public class MyClass {  
2     public static void main(String args[]) {  
3  
4         int x = 10;  
5         int y = 5;  
6         int z = 111;  
7         //Aqui habia otras cosas de char's  
8  
9     switch(z){  
10        case 111:  
11            System.out.println("Todo bien");  
12        case x:  
13            System.out.println("valor de x" + x);  
14            break;  
15        case y:  
16            System.out.println("valor de y" + y);  
17            break;  
18    }  
19 }  
20 }  
21 }
```

No pueden ir variables en los CASE

Ejercicio 4

¿Qué pasa con el siguiente código?

1. Which of the following Java operators can be used with boolean variables? (Choose all that apply)
 - A. ==
 - B. +
 - C. --
 - D. !
 - E. %
 - F. <=

Respuesta: A y D

Ejercicio 5

¿Qué pasa con el siguiente código?

Which two will compile, and can be run successfully using the following command? *

java Fred1 hello walls

class Fred 1{ public static void main(String args) { System.out.println(args[1]); } }

class Fred1 { public static void main(String[] args) { System.out.println(args); } }

class Fred1 { public static void main(String[] args) { System.out.println(args[1]); } } X

abstract class Fred1 { public static void main(String[] args) {System.out.println(args[2]);} }

Respuesta correcta

class Fred1 { public static void main(String[] args) { System.out.println(args); } }

Ejercicio 6

¿Qué pasa con el siguiente código?

Which three implementations are valid? *

1 punto

```
interface SampleCloseable {  
    public void close() throws java.io.IOException;  
}  
  
 class Test implements SampleCloseable { public void close() throws  
    java.io.IOException { // do something } }  
  
 class Test implements SampleCloseable { public void close() throws Exception { //  
    do something } }  
  
 class Test implements SampleCloseable { public void close() throws  
    FileNotFoundException { // do something } }  
  
 class Test extends SampleCloseable { public void close() throws java.io.IOException  
    { // do something } }  
  
 class Test implements SampleCloseable { public void close() { // do something } }
```

Las respuestas correctas son:

Primera opción: `public void close() throws java.io.IOException.`

Tercera opción: `public void close() throws FileNotFoundException.`

Última opción: `public void close()` (**sin throws**).

`public void close() throws Exception`

Inválido: `Exception` es más amplia que `IOException`, lo que viola la regla de excepciones en sobrescritura.

Ejercicio 7

Which two possible outputs? *

1 punto

```
public class Main {  
    public static void main(String[] args) throws Exception {  
        doSomething();  
    }  
    private static void doSomething() throws Exception {  
        System.out.println("Before if clause");  
        if (Math.random() > 0.5) { throw new Exception();}  
        System.out.println("After if clause");  
    }  
}
```

- Before if clause Exception in thread "main" java.lang.Exception at Main.doSomething (Main.java:21) at Main.main (Main.java:15).
- Before if clause Exception in thread "main" java.lang.Exception at Main.doSomething (Main.java:21) at Main.main (Main.java:15) After if clause.
- Exception in thread "main" java.lang.Exception at Main.doSomething (Main.java:21) at Main.main (Main.java:15).
- Before if clause After if clause.

Respuestas correctas:

- Primera opción: "**Before if clause Exception in thread "main" java.lang.Exception at Main.doSomething(Main.java:21) at Main.main(Main.java:15).**"
- Última opción: "**Before if clause After if clause.**"

Ejercicio 8

What is the result? *

1 punto

```
import java.text.*;
public class Align {
    public static void main(String[] args) throws ParseException{
        String[] sa = {"111.234", "222.5678"};
        NumberFormat nf = NumberFormat.getInstance();
        nf.setMaximumFractionDigits(3);
        for (String s : sa){ System.out.println(nf.parse(s));}
    }
}
```

- 111.234 222.5678
- An exception is thrown at runtime.
- 111.234 222.568
- 111.234 222.567

Respuesta correcta:

Primera opción: 111.234 222.568

Diferencia entre **parse** y **format**:

1. **parse**:

- Este método convierte una cadena de texto (como "111.234") a un objeto **Number** (como un **Double** o **Long**) sin modificar el valor.
- No aplica límites en los dígitos decimales, ya que su propósito es interpretar los datos exactamente como están en la entrada.

2. **format**:

- Este método convierte un objeto **Number** en una cadena de texto (**String**) aplicando las reglas de formato configuradas (como el número máximo de dígitos fraccionarios).

- Es aquí donde el límite impuesto por `setMaximumFractionDigits(3)` tiene efecto. Si llamas a `nf.format(number)` después de interpretar un número, verías que se aplican los límites de dígitos decimales.

Ejercicio 9

¿Qué pasa con el siguiente código?

```
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;
```

Respuestas:

A. 0

B. 30

C. 60

Ejercicio 10

The following are the complete contents of TestClass.java file. Which packages are automatically imported?

```
class TestClass {  
    public static void main(String[] args) {  
        System.out.println("hello");  
    }  
}  
  
java.util  
  
System  
  
java.lang  
  
java.io  
  
String
```

The package with no name

Ejercicio 11

¿Qué está mal en este código?

```
public static void main(String[] args) {  
    System.out.println("Hola Mundo");  
    // int [] arr = new int[]; // no compila porque no se indicó la longitud  
    //int [][] d = new int[];// no compila  
    int a [][] = new int[7][6]; //compila  
    Object [[[ objects = new Object[4][0][5]; //compila  
}}}
```

Ejercicio 12

¿Qué imprime este código?

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

Resultado:

2

1

Ejercicio 13

Qué se va a imprimir al ejecutar el método main()

```
public class Spider {  
    void spider(int a){  
        System.out.print("Spider");  
    }  
}  
  
public class Genero {  
    void genero(double b){  
        System.out.print("Genero");  
    }  
}  
  
public class SpiderMain {  
    public static void main(String[] args) {
```

```
Spider spider = new Spider();

Genero genero = new Genero();

spider.spider(4);

genero.genero(9.0);

}

}
```

Respuesta Spider Genero

Ejercicio 14

Con cuál de línea imprimirá Equals

```
//boolean r1 = a.equals(b);

//boolean r2 = a==b;

//boolean r3 = a.equalsIgnoreCase(b);

String a = "Hola";

String b = "holA";

boolean resultado = false;

if (.....){

    System.out.println("Equals");

}else {

    System.out.println("Not Equals");

}
```

Respuesta boolean r3 = a.equalsIgnoreCase(b);

Ejercicio 15

¿Qué sale en el siguiente código?

```
class One {  
    void foo() {  
        // Método en la clase base  
        System.out.println("Clase One: foo()");  
    }  
}  
  
class Two extends One {  
    // Opción 1: Misma visibilidad y tipo de retorno  
    @Override  
    void foo() {  
        System.out.println("Clase Two: foo() - package-private");  
    }  
  
    // Opción 2: Modificador más permisivo (public)  
    @Override  
    public void foo() {  
        System.out.println("Clase Two: foo() - public");  
    }  
  
    // Opción 3: Modificador más permisivo (protected)  
    @Override  
    protected void foo() {  
        System.out.println("Clase Two: foo() - protected");  
    }  
}
```

- ✓ Which three methods, inserted individually at line, will correctly complete class Two(Choose three)? *1/1

```
10. class One {  
11.     void foo() {}  
12. }  
13. class Two extends One {  
14.     //insert method here  
15. }
```

- int foo() /* more code here */ ✓
- void foo() /* more code here */
- public void foo() /* more code here */ ✓
- private void foo() /* more code here */
- protected void foo() /* more code here */ ✓

La primera opción int foo(){} <- esta mal porque no se puede sobrescribir regresando un tipo de dato si el metodo no lo define, aparte es un int aqui no existe la covarianza con tipos primitivos pero si fuera un Wrapper si hay covarianza y podriamos regresar por ejemplo un Integer si es que el metodo que define la herencia es un Number,

Las tres respuestas correctas son:

Segunda opción
Tercer opción
Cuarta opción

Ejercicio 16

¿Qué sale en el siguiente código?

```
public void suma(int...g,int []q){} //no compila porque los varargs debe ir al final de la lista de  
parametros
```

```
public void div(int[], ...int r){} //no compila porque los varargs van después del tipo de dato
```

```
public int suma(int..d,float...o){}
```

```
return d+o;
```

```
//no compila porque no sabe que es la variable d de los parámetros si se desea agregar  
varargs deben ser tres puntos*/
```

```
public void t (int...h, double...t){}//no compila porque solo se permite una vez usar varargs
```

```
public void ad(int b,double s, String...a){}// si funciona
```

```
public void ad(int b,double s, String ... a){// si funciona con espacio en el String y en el a
```

```
public void ad(int b,double s, String ... a){// no funciona dejando solo un espacio entre los  
puntos, deben ir juntos.
```

Ejercicio 17

¿Qué sale en el siguiente código?

```
public class VariableFinal {  
    public class Persona extends Carro {  
        final String nombre;  
        public Persona() {  
            Carro carro = new Carro();  
            nombre = "andres";  
        }  
    }  
    public class Carro {  
    }  
    public static void main(String[] args) {  
        VariableFinal v = new VariableFinal();  
        Carro c = v.new Carro();  
        //Persona persona = new Persona(); // error aqui  
    }  
}
```

Ejercicio 18

¿Qué sale en el siguiente código?

```
public static void main(String[] args) {  
    boolean x = false;  
    boolean z = false;  
    int y= 20;  
    System.out.println(x);  
    //true false |  
    x = (y!=10) ^ (z!=false);  
    System.out.println(x + " " + y + " " + z);  
}
```

Preguntas que vinieron APX.docx

Ejercicio 1

Valid argument for catch clause:

- A. **Throwable**
- B. Exception but NOT including RuntimeException
- C. CheckedException
- D. **RunTimeException**
- E. **Exception**

Recordar:

Tipos de datos permitidos en un `switch` en Java:

1. **Primitivos aceptados:**
 - o `byte`
 - o `short`
 - o `int`
 - o `char`
2. **Tipos no primitivos aceptados:**
 - o `String` (desde Java 7).
 - o `Enumeraciones (enum)`.
3. **Primitivos no aceptados:**
 - o `long, float, y double` no son permitidos como expresión de control en un `switch`.

Ejercicio 2

Asume que este switch acepta datos de tipo entero. ¿Qué tipo de dato no se podría colocar en el espacio en blanco?

```
public class App {  
    public static void main(String[] args) {  
        --- dayOfWeek = 1;  
        switch (dayOfWeek) {  
            case 1:  
                System.out.println("Monday");  
            case 2:  
                System.out.println("Tuesday");  
            case 3:  
                System.out.println("Other day");  
                break;  
        }  
    }  
}
```

- A. int
- B. long**
- C. char
- D. None of the above

Ejercicio 3

¿Qué se puede colocar al lado del case en un switch?

```
int dayOfWeek = 1  
switch (dayOfWeek) {  
    case dayOfWeek: // Error: `case` no puede usar una variable.  
        System.out.println("Monday");  
    case "HOLA": // Error: El tipo de "HOLA" (String) no coincide con el tipo de `dayOfWeek`.  
        System.out.println("Tuesday");  
    case 3:  
        System.out.println("Other day");  
        break;  
}
```

Ejercicio 4

¿Qué se puede colocar al lado del case en un switch?

```
final int dayOfWeek = 1
final Integer day = 1
switch (dayOfWeek) {
    case dayOfWeek: // Aceptado: `case` puede usar una variable constante si es final.
        System.out.println("Monday");
    case "HOLA": // Error: El tipo de "HOLA" (String) no coincide con el tipo de `dayOfWeek` .
        System.out.println("Tuesday");
    case day: // Error: una variable wrapper final no puede asignarse sólo los primitivos
        System.out.println("Other day");
        break;
}
```

Ejercicio 5

Elige todas las excepciones no verificadas

- a. **IllegalArgumentException**
- b. **NumberFormatException**
- c. IOException
- d. **NullPointerException**
- e. Exception
- f. **ArrayIndexOutOfBoundsException**

Ejercicio 6

¿Cuál es la salida?

```
public class Movie {  
    public static void main(String[] args) {  
        String movie = "Thriller";  
        switch (movie) {  
            case "Thriller":  
                System.out.println("Movie Thriller");  
            case "Comedy":  
                System.out.println("Movie Comedy");  
            case "Romance":  
                System.out.println("Movie Romance");  
                break;  
            case "Action":  
                System.out.println("Movie Action");  
        }  
    }  
}
```

- a. Movie Thriller
- b. Movie Thriller Movie Comedy
- c. **Movie Thriller Movie Comedy Movie Romance**

Ejercicio 7

¿Cuál es el resultado?

```
public class Movie {  
  
    public static void main(String[] args) {  
        String s = "ABCD";  
        s.trim();  
        s.toUpperCase();  
        s += " 123";  
        System.out.println(s.length());  
    }  
}
```

- a. **8**
- b. 4
- c. 7

Ejercicio 8

Usando **StringBuilder**, elige las opciones que den el mismo resultado que la concatenación original. (Elige todas las opciones aplicables).

```
public class Cadena {  
    public static void main(String[] args) {  
        String name = "Joe";  
        int age = 31;  
        String result = "My name is " + name + ", my age is " + age;  
        System.out.println(result);  
        StringBuilder sb = new StringBuilder();  
    }  
}
```

- a. **sb.append("My name is " + name + ", my age is " + age);**
- b. sb.insert("My name is " + name).append(", my age is "+ age);
- c. sb.insert("My name is").insert(name).insert(", my age is ").insert(age);
- d. **sb.append("My name is ").append(name).append(", my age is ").append(age);**

Ayuda con el contrato de equals:

De acuerdo con el contrato de `Object.equals`:

- El método debe ser reflexivo: un objeto debe ser igual a sí mismo (`x.equals(x)` debe devolver `true`).
- El método debe ser simétrico: si `x.equals(y)` es `true`, entonces `y.equals(x)` también debe ser `true`.
- El método debe ser transitivo: si `x.equals(y)` y `y.equals(z)` son `true`, entonces `x.equals(z)` también debe ser `true`.
- El método debe ser consistente: múltiples llamadas a `x.equals(y)` deben devolver el mismo resultado.
- Comparar con `null` debe devolver `false`.

Ejercicio 9

¿Qué afirmación es correcta sobre el siguiente código?

```
public class MyStuff {  
    String name;  
    MyStuff(String n) {  
        name = n;  
    }  
  
    public static void main(String[] args) {  
        MyStuff m1 = new MyStuff("guitar");  
        MyStuff m2 = new MyStuff("tv");  
        System.out.println(m2.equals(m1));  
    }  
  
    public boolean equals(Object o) {  
        MyStuff m = (MyStuff) o;  
        if (m.name != null) {  
            return true;  
        }  
        return false;  
    }  
}
```

- The output is true and MyStuff fulfills the `Object.equals()` contract
- The output is false and MyStuff fulfills the `Object.equals()` contract
- The output is true and MyStuff does NOT fulfill the `Object.equals()` contract.**
- The output is false and MyStuff does NOT fulfill the `Object.equals()` contract.

Ejercicio 10

¿Cuál es el resultado?

```
public class Bucle {  
    public static void main(String[] args) {  
        int[] at = {1, 2};  
        for (int x : at) {  
            System.out.println(x + ", ");  
            if (x < at.length) {  
                break;  
            }  
        }  
    }  
}
```

- a. 1, 2,
- b. Nocompila
- c. 1,
- d. 1, 2

Ejercicio 11

¿Cuál es el resultado?

```
class Control {  
    public static void main(String[] args) {  
        int i = 42;  
        String s = (i < 40) ? "Greater than" : false;  
        System.out.println(s);  
    }  
}
```

- a. Error de compilación
- b. Greater then
- c. Greater then false
- d. false

Ejercicio 12

¿El siguiente código compila?

```
public class estructura {  
    public static void main(String[] args) {  
        int num = 1;  
        switch (num) {  
            case 1:  
            case 2:  
                System.out.println("Caso 2");  
            case 3:  
                System.out.println("Caso 3");  
                Break;  
        }  
    }  
}
```

No compila porque el Break está con mayúscula y debería ser en minúsculas, si tuviera la b minúsculas compila sin problemas.

Ayuda:

Reglas para métodos que no retornan ningún valor:

1. Los métodos que no retornan un valor deben estar declarados con el tipo de retorno **void**.
2. **Opciones válidas dentro de un método void:**
 - Puedes usar un **return**; sin ningún valor para salir del método.
 - Puedes omitir completamente la palabra clave **return** si el método no necesita terminar de forma explícita.

Ejercicio 13

¿Cuál de los siguientes códigos podría estar presente en un método que no retorna ningún valor? (Elige dos opciones)

- a. return null;
- b. return void;
- c. **return;**
- d. **Omitir el uso de la palabra clave return**

Ejercicio 14

_____ occurs when a subclass redefines a method from its superclass, while
_____ happens when multiple methods in the same class share the same name

- but differ in their parameters.
- a. Overloading / Overriding
 - b. **Overriding / Overloading**
 - c. Inheriting / Overriding
 - d. Overloading / Inheriting

APX

Ejercicio 15

Cuáles son los dos tipos de step que existen Batch APX

- a. Creator, Processor
- b. Processor, Task
- c. Chunk, Reader
- d. **Chunk,Task**

Ejercicio 16

De la lista siguiente, indique cuales son las invocaciones restringidas en la Arquitectura Batch:

- a. **Invocación WebServices, servicios REST**
- b. **Acceso a recursos externos vía HTTP**
- c. **Uso de Pre y Post Acciones**
- d. Escritura en Registro de Operaciones

e. Escritura en el Diario Electrónico

Ejercicio 17

¿Qué nivel de profundidad es permitido al realizar invocaciones de librería a librería?

a. Profundidades a 3 niveles: Transacción > Librería > Librería > Librería

b. Profundidades a 5 niveles: Transacción > Librería > Librería > Librería > Librería > Librería

c. Profundidades a 7 niveles: Transacción > Librería > Librería > Librería > Librería > Librería > Librería > Librería

d. Profundidades a 9 niveles: Transacción > Librería > Librería

Ejercicio 18

Se Debe evitar en una transacción o biblioteca, dentro de su lógica, invocar más de _____ librerías a la vez

a. 7

b. 15

c. 5

d. 9

Ejercicio 19

Está permitido que la aplicación modifique o informe una variable de entorno de la JVM

a. Verdadero

b. Falso

Ejercicio 20

El acceso a los datos debe realizarse utilizando las utilidades que Arquitectura proporciona desde el asistente de biblioteca. En el caso de bases de datos relacionales, ¿Qué utilidad se debe usar?

- a. APIConector
- b. Elastic
- c. Noconozco la respuesta
- d. Datio
- e. JDBC**

Ejercicio 21

Este patrón suele representarse en una librería que contiene un solo método “execute” con parámetros que actúan como filtros y que engloba diversas funcionalidades dispares en función de sus parámetros de entrada

- a. Contenedor Mágico**

- b. CRUD
- c. Paginacion
- d. Blob

Ejercicio 22

La información utilizada en la lógica de negocio de la transacción debe incluirse en _____.

- a. Los archivos locales.
- b. Las variables globales.
- c. Los parámetros de entrada.**
- d. Las variables locales