

Ejercicios

1. Polimorfismo y Excepciones

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

2. Hilos (Threads)

Considera el siguiente bloque de código:

```
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(); - Los hilos se ejecutan con un start() Si se llama run() directamente ya no son hilos  
        t2.start(); - Los hilos se ejecutan aleatoriamente  
    }  
}
```

¿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

3. Listas y Excepciones

Considera el siguiente bloque de código:

```
import java.util.ArrayList;  
import java.util.List;  
  
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 ----- Itera sin problemas 3 veces despues que se quede sin indices en la lista, entra Catchen función e imprime la cadena de String
2. 1 2 3
3. Exception caught
4. 1 2 3 4

4. Herencia, Clases Abstractas e Interfaces

Considera el siguiente bloque de código:

```

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 es de tipo Vehicle pero apunta a un Car
        myCar.fuel(); Lama al método fuel de la clase Car
        ((Movable) myCar).move(); Realiza un cast para llamar al método 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
4. Runtime exception

5. Polimorfismo y Sobrecarga de Métodos

Considera el siguiente bloque de 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

- Las clases abstractas son plantillas que pueden contener métodos concretos y abstractos

- Una clase que extiende una clase abstracta debe implementar todos sus métodos abstractos

- Las interfaces, son contratos que solo contienen métodos abstractos

- Una clase que implementa una interfaz debe proporcionar implementación para todos los métodos definidos en ella

- Si un objeto es de una clase que implementa una interfaz puede convertirse explícitamente al tipo de esa interfaz para acceder a sus métodos.

6. Hilos y Sincronización

Considera el siguiente bloque de código:

```
class Counter {
    private int count = 0;

    public synchronized void increment() { Synchronized permite que entre un hilo, y después otro hilo  
        count++; Sin Synchronized no hay garantía que llegue a 2000
    }

    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()); El join() le dice a main que los hilos deben ejecutarse primero, de lo  

        contrario println se podría adelantar en la ejecución
    }
}
```

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

1. 2 00 0
2. 1000
3. Variable count is not synchronized
4. Compilation error

7. 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");  
    }  
}
```

- La sobrescritura se realiza usando el mismo nombre, firma y tipo de retorno del método en la clase padre.

```
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();  
        }  
    }  
}
```

-La lista animals contiene objetos de las clases Dog, Cat y Animal, pero todos son tratados como Animal debido a la referencia de tipo genérico List<Animal>.

¿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

8. Manejo de Excepciones y Herencia

Considera el siguiente bloque de código:

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

Un método sobrescrito puede:

1. Declarar ninguna excepción (si el método de la clase base no declara excepciones).
2. Declarar las mismas excepciones que el método en la clase base.
3. Declarar excepciones más específicas (subclases).
4. No puede declarar excepciones más generales o nuevas excepciones no verificadas.

- Derived sobrescribe el método show de Base. El método en Derived lanza una excepción más específica, FileNotFoundException, que es una subclase de IOException

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

1. Base show
2. Derived show

3. Exception caught
4. Compilation error

9. Concurrency y Sincronización

Considera el siguiente bloque de código:

```
class SharedResource {
    private int count = 0;  -- El valor inicial de count es 0.

    public synchronized void increment() {
        count++;           -----El hilo IncrementThread incrementa el valor de count 1000 veces, y
    }                     -----El hilo DecrementThread decrementa el valor de count 1000 veces.

    public synchronized void decrement() {
        count--;           --- Como los métodos increment y decrement están
    }                     sincronizados con la palabra clave synchronized, se garantiza
                           que solo un hilo pueda acceder al método en un momento
                           dado.

    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

10. 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 {

```

--- El método verifica si el objeto almacenado en item es una instancia de String con item instanceof String.

```

    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"); --- Se asigna un valor de tipo String

        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

11. ..

```

public class Main {
    public static void main(String[] args) {
        Padre objetoPadre = new Padre();
        Hija objetoHija = new Hija();
        Padre objetoHija2 = (Padre) new Hija();
    }
}

```

Objeto de clase padre, referencia de tipo padre
 objeto de clase hija, referencia clase hija
 objeto de clase hija, referencia tipo padre (upcasting)

```

    objetoPadre.llamarClase();
    objetoHija.llamarClase();
objetoHija2.llamarClase();

```

```

    Hija objetoHija3 = (Hija) new Padre();
    objetoHija3.llamarClase();
}
}

```

---- El método llamarClase() nunca se ejecuta en objetoHija3 porque el programa se detiene debido a la excepción antes de llegar a esa línea.

```

public class Hija extends Padre {
    public Hija() {
        // Constructor de la clase Hija
    }

    @Override
    public void llamarClase() {
        System.out.println("Llame a la clase Hija");
    }
}

public class Padre {
    public Padre() {
        // Constructor de la clase Padre
    }

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

```

Resultado:

```

1.  Llame a la clase Padre
    Llame a la clase Hija
    Llame a la clase Hija
    Error: java.lang.ClassCastException

```

2. Llame a la clase Padre
 - Llame a la clase Hija
 - Llame a la clase Hija
 - Llame a la clase Hija

3. Llame a la clase Padre
Llame a la clase Hija
Llame a la clase Hija
Llame a la clase Padre

12. ..

```
import java.text.NumberFormat;
import java.text.ParseException;
import java.util.Scanner;
import java.util.ArrayList;
import java.util.List;

public class Ejemplos {

    public static void main(String[] args) {
        Animal uno=new Animal(); -- Estos imports no se utilizan
        Animal dos=new Dog(); -- Se creo un objeto de la clase Dog y se ejecuta el método makeSound

        uno.makeSound();
        dos.makeSound();

        Dog tres=(Dog)new Animal(); ----- Se intenta realizar un downcasting: convertir un objeto de tipo
        tres.makeSound(); Animal a una referencia de tipo Dog. Esto genera una
                           ClassCastException en ejecución porque el objeto real es de tipo
                           Animal, no de tipo Dog
    }
}

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

3. Animal sound Wau Wau Animal sound

4. Animal sound

13....

```

import java.text.NumberFormat;
import java.text.ParseException;
import java.util.Scanner;
import java.util.ArrayList;
import java.util.List;
import java.lang.*;

```

--Estos imports no se utilizan en este código

```

public class Ejemplos {

```

```

    public static void main(String[] args) {

```

```

        Cambios uno=new Cambios();

```

Se crea objeto de la clase Cambios, referencia 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";

```

----- Los String y los primitivos son inmutables, por lo que no aplica los cambios.

```

        x=5;

```

```

    }

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

```

--- Integer aunque es wrapper es immutable, en cambio StringBuilder si es mutable, por lo que esta accion hace borrar la cadena

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

14. ...

```

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?
}

```

La interfaz i2 extiende de i1. Esto significa que i2 hereda el método m1 de i1, cualquier clase que implemente i2 debe proporcionar implementaciones para ambos métodos, m1 y m2.

Debido a la relación de herencia entre las interfaces, implementar i2 ya implica cumplir con los requisitos de i1.

1. solo m1
2. m1 y m2
3. ninguno
4. error compilaci

15.

```
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(); ---el método makeSound de la clase Dog será invocado debido al polimorfismo.
        try {
            myDog.makeSound(); El bloque try-catch nunca captura una excepción, ya que el método
        } catch (Exception e) { ejecutado (Dog.makeSound) no lanza ninguna
            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

16.

```
import java.util.*;
import java.lang.*;
import java.io.*;
```

```
class Main {
```

```
public static void main(String[] args) {  
    String str = "1a2b3c4d5e6f";  
    String []splitStr = str.split("//D");  
  
    for(String elemento : splitStr){  
        System.out.println(elemento);  
    }  
}
```

SALIDA DE CONSOLA:1a2b3c4d5e6f