Ejercicio 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

El código no compilará debido a que el método makeSound en Dog no cumple con la regla de excepciones para la sobrescritura. El compilador detectará un error porque RuntimeException no es una excepción que se ajusta a la regla de sobrescritura de excepciones del método makeSound en la clase base Animal.

****

Ejercicio 2

Ejercicio de 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();
        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

Ejecución Paralela: Los métodos run() de los hilos t1 y t2 se ejecutan en paralelo, lo que significa que ambos hilos están trabajando al mismo tiempo.

Impresión: Cada hilo imprimirá "Thread is running" una vez.

****

Ejercicio 3

Ejercicio de 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

2- 1 2 3

3- Exception caught

4- 1 2 3 4

Lista numbers:

Se crea una lista de enteros y se añaden tres elementos: 1, 2, y 3.

Bucle for:

El bucle for itera desde i = 0 hasta i <= numbers.size().numbers.size() es 3, por lo que el bucle intentará acceder a numbers.get(3) en la última iteración (i = 3).

Acceso a la Lista:

Las posiciones válidas en la lista son 0, 1, y 2.

Intentar acceder a numbers.get(3) causa una IndexOutOfBoundsException porque el índice 3 está fuera del rango válido.

Manejo de Excepciones:

La excepción IndexOutOfBoundsException es capturada por el bloque catch, que imprime "Exception caught

****

Ejercicio 4

Ejercicio de 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.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

4- Runtime exception

¿Cuál crees que es la respuesta correcta?

myCar.fuel();

Llama al método fuel() del objeto Car, que imprime "Car is refueled".

((Movable) myCar).move();

Realiza un cast de myCar (de tipo Vehicle) a Movable y llama al método move(). Esto es válido porque Car implementa Movable, y el cast es seguro. El método move() imprimirá "Car is moving".

****

Ejercicio 5

Ejercicio de 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

¿Cuál crees que es la respuesta correcta?

obj.display(5);

La llamada display(5) se corresponde con el método display(int num).

En la instancia de Child, se usa el método sobrescrito en Child, que imprime "Child: 5".

obj.display("Hello");

La llamada display("Hello") se corresponde con el método display(String msg).

En este caso, el método no está sobrescrito en Child, por lo que se utiliza el método display(String msg) de la clase Parent, que imprime "Parent: Hello".

****

Ejercicio 6

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

Ejecución de Hilos:

Ambos hilos (t1 y t2) incrementan el valor de count en la misma instancia de Counter. Dado que el método increment es synchronized, se asegura que solo un hilo puede ejecutar este método a la vez, evitando condiciones de carrera y garantizando que el valor de count se incremente de forma segura.

Resultado Final:

Cada hilo incrementa count 1000 veces, por lo que el valor final esperado de count es 2000 después de que ambos hilos hayan terminado su ejecución.

****

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

¿Cuál crees que es la respuesta correcta?

La lista animals contiene tres elementos: un Dog, un Cat, y un Animal.

Durante la iteración del bucle for, el método makeSound() se llama en cada objeto de la lista. Debido a la polimorfismo en Java, el método makeSound() que se ejecuta es el que corresponde al tipo real del objeto, no al tipo de la referencia.

Dog: Llama a makeSound() en un objeto Dog, que imprime "Bark".

Cat: Llama a makeSound() en un objeto Cat, que imprime "Meow".

Animal: Llama a makeSound() en un objeto Animal, que imprime "Animal sound".

****

Ejercicio 8

Ejercicio de 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");
        }
    }
}
```

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

1- Base show

2- Derived show

3- Exception caught

4- Compilation error

Sobrescritura y Excepciones: En Java, cuando un método de una subclase sobrescribe un método de una superclase, el método sobrescrito en la subclase puede lanzar excepciones que son una subclase de las excepciones lanzadas por el método de la superclase. En este caso, FileNotFoundException es una subclase de IOException, por lo que el método show() en Derived es una sobrescritura válida.

Llamada al Método show():

Aunque obj es de tipo Base, se refiere a una instancia de Derived. Debido al polimorfismo, se llama al método show() de Derived.

El método show() de Derived imprime "Derived show".

Manejo de Excepciones:

El bloque try en main está preparado para capturar una IOException. Dado que FileNotFoundException es una subclase de IOException, no se lanzará una excepción no controlada, y el bloque catch no se ejecutará.

****

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

Los métodos increment y decrement están sincronizados, por lo que las operaciones sobre count están protegidas contra condiciones de carrera.

Cada hilo incrementa o decrementa count 1000 veces. Dado que los métodos están sincronizados, el valor final de count después de que ambos hilos terminen su ejecución debería ser el resultado de 1000 incrementos y 1000 decrementos, es decir, el valor de count debería ser 0.

****

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

En este código, el método getItem() de la clase Box realiza un casting inseguro de item a String. Sin embargo, dado que en el caso de stringBox se está usando un Box<String> y se ha establecido un String como ítem, el casting es seguro y no se lanza ninguna excepción.

La ClassCastException se lanzaría solo si el tipo de item no fuera String, pero en este caso, el ítem es efectivamente un String, por lo que el método getItem() devuelve "Hello" sin lanzar ninguna excepción.}