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

Esto porque el método makeSound en la clase Dog tiene una firma distinta en cuanto a las excepciones que puede lanzar, en comparación con el método makeSound en la clase Animal.

4o

****

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)

Dado que los hilos 2 hilos creados se ejecutan de manera aleatoria y ambos imprimen System.out.println("Thread is running");

4- Compilation error

****

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

La razón es que el bucle for está intentando acceder a un índice fuera del rango de la lista.

****

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

La clase Car extiende la clase abstracta Vehicle e implementa la interfaz Movable. En el método main, el objeto myCar es de tipo Vehicle, pero como es una instancia de Car, el método fuel se ejecuta correctamente.

¿Cuál crees que es la respuesta correcta?

****

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?

Esto porque el objeto obj es de tipo Parent, pero se instancia como Child.

****

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

¿Cuál crees que es la respuesta correcta?

La clase Counter tiene un método increment que está sincronizado, lo que significa que solo un hilo puede acceder a él a la vez.

****

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 instancias de Dog, Cat y Animal. Cuando el bucle for itera sobre la lista y llama al método makeSound, se ejecuta el método correspondiente a cada instancia.

****

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

¿Cuál crees que es la respuesta correcta?

El error yace en que el método show en la clase Derived lanza una FileNotFoundException, que es una excepción comprobada

****

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

¿Cuál crees que es la respuesta correcta?

La respuesta es 0 porque los métodos increment() y decrement() de la clase SharedResource están sincronizados.

****

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

¿Cuál crees que es la respuesta correcta?

La respuesta es "Hello" debido a cómo se maneja la instancia y el tipo de la clase Box en combinación con el método getItem. A continuación, te explico el flujo del programa y por qué no se produce una excepción: