Preguntas propuestas

```
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- Excepcion caught
4- Compilation error

Dado que el método `makeSound` en `Dog` simplemente imprime "Dog barks" y no lanza ninguna excepción.

```
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

La salida esperada será "Thread is running" impresa dos veces, pero el orden en que se imprimen puede variar debido al comportamiento concurrente de los hilos.

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

El bucle hace que la ultima iteracion del bucle (3) se salga del numero de elementos que tiene la lista por lo que se lanza una Exception que es trabajada eh imprime “1 2 3 Exception caught”

```
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
`myCar.fuel();` llama al método `fuel` de la clase `Car`, que imprime "Car is refueled".
`((Movable) myCar).move();` realiza un casting de `myCar` a `Movable` y luego llama al método `move` de la interfaz `Movable`, que está implementado en la clase `Car` y que imprime "Car is moving".

```
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?
Child: 5 Parent: Hello

Cuando el método display(int num) es llamado con el argumento 5, el método sobrescrito en la clase Child se ejecuta ya que la instancia real del objeto es de tipo Child

Cuando el método display(String msg) es llamado con el argumento "Hello", el método correspondiente en la clase Parent se ejecuta ya que la clase Child no sobrescribe este método.

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

```
2000
```

Dado que el método increment está sincronizado, solo un hilo puede ejecutarlo a la vez. Esto viene condiciones de carrera y asegura que cada incremento de count sea seguro en términos de concurrencia.

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

Bark Meow Animal sound

La primera instancia es Dog, por lo que makeSound imprimirá "Bark".
La segunda instancia es Cat, por lo que makeSound imprimirá "Meow".
La tercera instancia es Animal, por lo que makeSound imprimirá "Animal sound".

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

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");
        }
    }
}
```

Derived show

En tiempo de ejecución, el método show de Derived será invocado debido al polimorfismo. FileNotFoundException es una subclase de IOException, la declaración throws IOException en Base permite que Derived lance FileNotFoundException.
No se lanza ninguna excepción durante la ejecución del método show por lo que el bloque catch no se ejecuta.

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

0

IncrementThread incrementará el contador 1000 veces.

DecrementThread decrementará el contador 1000 veces.

Ambos métodos increment() y decrement() están sincronizados lo que asegura que solo un hilo puede modificar count a la vez.

Después de que ambos hilos hayan terminado, el valor de count debería ser 0, ya que habrá 1000 incrementos y 1000 decrementos.

```
class Box<T> {
    private T item;

    public void setItem(T item) {
        this.item = item;
    }
```

```
    public T getItem() throws ClassCastException {
        if (item instanceof String) {
            return item; // Safe 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");
        }
    }
}
```

Hello

stringBox es una instancia de Box<String>, y se asigna la cadena "Hello" a item.
Al llamar a getItem(), el código verifica si item es una instancia de String, lo cual es verdadero en este caso.
El cast (T) item se realiza de manera insegura pero es válido aquí porque item ya es de tipo String.
No se lanza ninguna excepción, por lo tanto, el bloque catch no se ejecuta.