JAVA PROBLEMAS PROPUESTOS (PARTE I)

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

EXPLICACIÓN

La respuesta correcta es 1, a pesar de que el método que tiene lanza un Exception en el main se hace responsable de la excepción con el try/catch, ahora al invocar el método makeSound() por el principio de polimorfismo se invoca el método declarado en la clase Dog sin ninguna excepción por lo que el catch no tiene nada que atrapar.

****

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

EXPLICACIÓN

Se imprimirá dos veces porque son dos hilos con el mismo método, sin embargo el planificador de hilos del sistema operativo determina el orden en que los hilos se ejecutan, y esto puede variar entre diferentes ejecuciones del mismo programa. Por lo que la respuesta es la 3.

****

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

EXPLICACIÓN

El ciclo for se ejecuta correctamente hasta que se intenta acceder a una posición fuera de rango por lo que la excepción es atrapada después de haber impreso 1 2 3 y después se imprime el mensaje "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?

EXPLICACIÓN

La respuesta correcta es la 1, si observamos tenemos una variable de referencia de tipo Vehicle a la que se le asigna un objeto de la clase Car, y en la clase car que extiende de Vehicle tiene implementados los métodos de fuel() y el move() dados por Vehicle y la interfaz Movable, respectivamente. Ahora la referencia Vehicle conoce el método fuel() polo que lo puede invocar sin problema para imprimir “Car is refueled”, y después se aplica un cast a myCar para ser de tipo Movable por loque ahora conoce el método move() y se puede invocar para imprimir “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?

EXPLICACIÓN

La respuesta correcta es la 1, porque aplicando el principio de polimorfismo y herencia, se considera que java toma el método o variable de instancia más especifico es decir que los métodos de la clase del objeto tiene preferencia sobre los de la superclase, y mientras la variable de referencia tenga conocimiento de los métodos permitirá invocarlos, pero se comportarán de acuerdo al objeto.

Tomando en cuenta esto se invoca el método display(int), que esta sobrescrito en la clase Child por lo que este es el que se ejecuta, mientras que para el método display(String), mencionando que es un sobrecarga de display que recibe un String en vez de un int, esté método lo toma de la superclase ya que no lo tiene sobrescrito la subclase 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?

****

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?

EXPLICACIÓN

Por el principio de polimorfismo y herencia, podemos ver que cat y dog extienden de Animal lo que quiere decir que Animal y sus instancias pueden asignadas en variables de referencia tipo Animal lo que sucede en el Arraylist, ahora cuando se invoca el método makeSound(); java invoca el método más especifico es decir comienza por la clase del objeto y va subiendo a la superclase en busca del método. Como podemos esto hace reusable el código.

****

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?

EXPLICACIÓN

Lo primero a considerar es que tenemos un tipo de excepción checked, por lo que debemos tomar responsabilidad de ella, lo cual se hace con eltry/catch, ahora se invoca el método show(), toma el de de la clase del objeto por el principio de polimorfismo y dentro de el no se lanza ninguna excepción, y se ejecuta sin problemas el método imprimiendo “Derived show”.

****

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?

EXPLICACIÓN

Dado que ambos métodos increment() y decrement() están sincronizados, no habrá problemas de condiciones de carrera. Esto significa que cada incremento y decremento de count se realiza de manera segura. Como IncrementThread incrementa count 1000 veces y DecrementThread decrementa count 1000 veces, el valor final de count será 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

¿Cuál crees que es la respuesta correcta?

EXPLICACIÓN

El método getItem podría ser el que provoque una excepción, sin embargo al asignar el String “Hello” en la variable item cumple con la condición requerida el método por lo que se ejecuta sin excepciones y se imprime “Hello”.