Lizbeth Cortés León

Ejercicios propuestos APX

Las respuestas correctas están marcadas con amarillo.

Ejercicio 1

Tema: 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?

Opciones:

1.- Dog barks
2.- Animal makes a sound
3.- Exception caught
4.- Compilation error

Justificación:

El método `makeSound()` en la subclase `Dog` no puede lanzar excepciones más generales que el método en la superclase `Animal`. La declaración de excepciones en el método sobrescrito debe ser más específica o igual que la del método original en la superclase.

Ejercicio 2

Tema: 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?

Opciones:

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

Justificación:

Al ejecutar el código, se crean dos hilos (t1 y t2) que ejecutan la misma clase MyThread. Cada hilo imprimirá "Thread is running" cuando se ejecute. Como ambos hilos comienzan a ejecutarse casi simultáneamente, es posible que ambos impriman el mensaje al mismo tiempo o en un orden que varíe en cada ejecución. Por lo tanto, en la consola se mostrará el mensaje "Thread is running" impreso dos veces, pero el orden en que aparece puede ser diferente en cada ejecución del programa.

Ejercicio 3

Tema: 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?

Opciones:

1.- 1 2 3 Exception caught
2.- 1 2 3
3.- Exception caught
4.- 1 2 3 4

Justificación:

Al ejecutar el código, se crea una lista numbers que contiene los valores 1, 2 y 3. El bucle intenta acceder a los elementos de la lista desde el índice 0 hasta el índice 3, que está fuera del rango válido. Al llegar al índice 3, se lanza una excepción IndexOutOfBoundsException, la cual es capturada por el bloque catch. Por lo tanto, en la consola se imprimen los números 1, 2 y 3, seguidos del mensaje "Exception caught".

Ejercicio 4

Tema: 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?

Opciones:

1.- Car is refueled Car is moving
2.- Car is refueled
3.- Compilation error
4.- Runtime exception

Justificación:

Al ejecutar el código, se crea un objeto de la clase Car y se asigna a una variable de tipo Vehicle. Primero, se llama al método fuel() del objeto, que imprime "Car is refueled". Luego, se realiza un cast para tratar el objeto como una instancia de Movable y se llama al método move(), lo que imprime "Car is moving".

La interfaz Movable define el método move(), pero no proporciona una implementación. La clase Car implementa la interfaz Movable y proporciona su propia implementación del método move(). Cuando se realiza la conversión de tipo (casting) de myCar a Movable y se llama al método move(), la implementación que se invoca es la de la clase Car, porque myCar es una instancia de Car.

Por lo tanto, en la consola se verá primero "Car is refueled" y luego "Car is moving"

Ejercicio 5

Tema: 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?

Opciones:

1.- Child: 5 Parent: Hello
2.- Parent: 5 Parent: Hello
3.- Child: 5 Child: Hello
4.- Compilation error

Justificación:

Al ejecutar el código, se crea un objeto de la clase Child pero se le asigna a una variable de tipo Parent. Cuando se llama al método display con un entero (5), se invoca la versión del método en la clase Child, ya que es una sobrecarga, y se imprime "Child: 5".

Cuando se llama al método display con una cadena de texto ("Hello"), se usa la versión del método en la clase Parent, ya que Child no sobrescribe este método, y se imprime "Parent: Hello". Por lo tanto, la salida en la consola es "Child: 5" seguido de "Parent: Hello".

Ejercicio 6

Tema: 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?

Opciones:

1.- 2000

2.- 1000
3.- Variable count is not synchronized
4.- Compilation error

Justificación:

En este código, dos hilos están incrementando un contador compartido. El método increment() está sincronizado, lo que significa que solo un hilo puede incrementar el contador a la vez, evitando errores. Cada hilo incrementa el contador 1000 veces, así que después de que ambos hilos terminen, el contador debe ser 2000. Por lo tanto, la salida en la consola es 2000.

Ejercicio 7

Tema: 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?

Opciones:

1.- Animal sound Animal sound Animal sound
2.- Bark Meow Animal sound
3.- Animal sound Meow Bark
4.- Compilation error

Justificación:

Se crea una lista de animales que incluye un Dog, un Cat y un Animal. Al recorrer la lista y llamar al método makeSound() de cada objeto, se ejecuta el método específico de cada tipo de animal. Así, el Dog imprime "Bark", el Cat imprime "Meow", y el Animal imprime "Animal sound".

Ejercicio 8

Tema: 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?

Opciones:

1.- Base show
2.- Derived show
3.- Exception caught
4.- Compilation error

Justificación:

Al intentar compilar el código, se produce un error porque el método show() en la clase Derived lanza una excepción más específica (FileNotFoundException) en lugar de una excepción más general (IOException) como el método show() en la clase Base. Debido a que esto no es permitido, resulta en un error de compilación.

Ejercicio 9

Tema: 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?

Opciones:

1.- 1000
2.- 0
3.- -1000
4.- Compilation error

Justificación:

En este ejercicio, dos hilos trabajan con el mismo recurso compartido: uno incrementa el valor y el otro lo decrementa, ambos 1000 veces. Los métodos que modifican el valor están sincronizados, lo que asegura que no haya conflictos entre los hilos. Después de que ambos hilos terminen su ejecución, el valor del recurso compartido regresa a 0, ya que el incremento y el decremento se cancelan entre sí.

Ejercicio 10

Tema: 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?

Opciones:

1.- Hello
2.- Exception caught
3.- Compilation error
4.- ClassCastException

Justificación:

Aquí se usa una caja genérica para almacenar una cadena de texto ("Hello"). El método getItem() verifica si el contenido de la caja es una cadena, lo cual es cierto en este caso. Como el contenido es una cadena, se imprime "Hello" en la consola sin errores.

Ejercicio 11

Tema: Casteo de clases y herencia

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

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

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");
     }
 }
```

Opciones:

a) Llame a la clase Padre
Llame a la clase Hija

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

b) Llame a la clase Padre
Llame a la clase Hija
Llame a la clase Hija
Llame a la clase Hija

c) Llame a la clase Padre
Llame a la clase Hija
Llame a la clase Hija
Llame a la clase Padre

d) No se UnU

Justificación:

- **objetoPadre.llamarClase();** imprime "Llame a la clase Padre" porque objetoPadre es una instancia de Padre.
- **objetoHija.llamarClase();** imprime "Llame a la clase Hija" porque objetoHija es una instancia de Hija.
- **objetoHija2.llamarClase();** también imprime "Llame a la clase Hija" porque objetoHija2 es en realidad una instancia/objeto de tipo Hija, aunque se referencie como Padre.
- **Hija objetoHija3 = (Hija) new Padre();** intenta convertir directamente un objeto Padre a Hija, lo cual es incorrecto y causa una excepción ClassCastException.

Ejercicio 12

Tema: Casteo de clases y herencia

De acuerdo al siguente código:

```
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();
      Animal dos=new Dog();

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

     Dog tres=(Dog)new Animal();
      tres.makeSound();
     }
}

class Animal {
      void makeSound() {
          System.out.println("Animal sound");
      }
   }

  class Dog extends Animal {
     void makeSound() {
       System.out.println("Wau Wau");
     }
   }
```

¿Cuál es el resultado de ejecutar el siguiente código?

Opciones:

1) Animal sound Wau Wau compilation error
2) Compilation Error
3) Animal sound Wau Wau Animal sound
4) Animal sound

Justificación:

- **uno.makeSound();** imprime "Animal sound" porque uno es una instancia de Animal.
- **dos.makeSound();** imprime "Wau Wau" porque dos es una instancia de Dog, aunque se refiere como Animal.
- **Dog tres = (Dog) new Animal();** Aquí se intenta hacer un cast de un objeto Animal a Dog, lo cual no es válido en Java porque no todos los objetos Animal son instancias de Dog. Esto causa una excepción ClassCastException en

tiempo de ejecución, y también un error de compilación ya que el cast es incompatible. Para hacer el cast de forma correcta tendría que escribirse **Dog tres = (Dog) dos;**

Ejercicio 13

De acuerdo al siguiente código:

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

public class Ejemplos {


   public static void main(String[] args) {


     Cambios uno=new 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";
          x=5;
      }


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

¿Cuál es el resultado?

Opciones:

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

Justificación:

Esto es lo que sucede al ejecutar el código:

1. **uno.makeSound(x, hola);**:
   - x es un tipo primitivo (int) que se pasa por valor, por lo que el cambio a x dentro del método no afecta al valor original de x en main.
   - hola es un String, que en Java es inmutable. El cambio a s dentro del método no afecta al valor original de hola.
2. **uno.makeSound(x2, hola2);**:
   - x2 es un objeto Integer, pero se pasa por valor. El cambio a x dentro del método no afecta a x2 en main.
   - hola2 es un StringBuilder, que es mutable. El método s.delete(0, s.length()) vacía el contenido de hola2, así que hola2 se convierte en una cadena vacía.
3. **Salida final**:
   - x y hola permanecen como 1 y "hola", respectivamente.
   - x2 sigue siendo 4.
   - hola2 se vacía y se convierte en una cadena vacía.

Por eso, la salida es "Cambios?: 1,hola,4,".

Ejercicio 14

De acuerdo al siguiente código:

```
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?
}
```

Opciones:

1) solo m1
2) m1 y m2
3) ninguno
4) error compilación

Justificación:

En la clase animal, que implementa las interfaces i1 e i2, se debe implementar todos los métodos de las interfaces.

La interfaz i2 extiende i1, por lo que i2 hereda el método m1 de i1 y agrega su propio método m2. Entonces, la clase animal debe implementar ambos métodos:

- **m1** (heredado de i1)
- **m2** (específico de i2)

Ejercicio 15:
De acuerdo al siguiente código:

```
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);
        }
    }
}
```

¿Qué se obtiene como resultado? Justificación

El código tiene un error en la línea donde se usa el método split. El patrón //D no es válido. El patrón correcto para dividir una cadena por caracteres no numéricos es \D, no //D.

Con la corrección, el código se vería así: str.split("\\D"). Esto dividirá la cadena "1a2b3c4d5e6f" en partes usando caracteres no numéricos como delimitadores. La salida del programa será:

1 2 3 4 5 6

Esto ocurre porque split("\\D") divide la cadena en cada lugar donde aparece un carácter no numérico, dejando solo los dígitos.