Nombre: Gina Belén Molina Gutiérrez
Semana: 3

Ejercicio 1

Polimorfismo y Excepciones:

- El polimorfismo es la capacidad de una variable, función o objeto para tomar múltiples formas.
- En Java, permite que una referencia de clase base apunte a objetos de clase derivada.
- Las excepciones son eventos que interrumpen el flujo normal de ejecución de un programa.
- Se utilizan para manejar errores y otras condiciones excepcionales de manera controlada.

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

En la clase `Dog`, el método `makeSound` está sobrescribiendo (override) el método `makeSound` de la clase `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

Aquí está el análisis de por qué esa es la salida:

1. La clase `MyThread` extiende la clase `Thread` y sobrescribe el método `run` para imprimir "Thread is running".
2. En el `main` method, se crean dos instancias de `MyThread`, `t1` y `t2`.

3. Las dos instancias se inician llamando a `start()`, lo que provoca que ambos hilos ejecuten su método `run`.Por lo tanto, la salida será "Thread is running" impreso dos veces, pero el orden en que se imprimen puede variar en diferentes ejecuciones.

****

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 lista `numbers` tiene tres elementos, por lo que los índices válidos son 0, 1 y 2. Cuando `i` es 3, se intentará acceder a `numbers.get(3)`, lo que provocará una excepción `IndexOutOfBoundsException` porque no hay un cuarto elemento en la lista.

Cuando se lanza la excepción `IndexOutOfBoundsException`, se captura en el bloque `catch` y se 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");
   }// Es correcto

   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?
1- Car is refueled Car is moving

Sí, el código es correcto. Define una interfaz Movable con un método move(), una clase abstracta Vehicle con un método abstracto fuel(), y una clase Car que extiende Vehicle e implementa Movable. En la clase Main, creas una instancia de Car, la usas como Vehicle para llamar al método fuel(), y luego haces un cast a Movable para llamar al método move().

****

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?

1- Child: 5 Parent: Hello

Sobrecarga de métodos (Method Overloading):

- Ocurre cuando en una misma clase hay más de un método con el mismo nombre pero con diferentes parámetros (diferente lista de argumentos).
- En la clase Parent, hay dos métodos display sobrecargados:

```
void display(int num) {
    System.out.println("Parent: " + num);
}

void display(String msg) {
    System.out.println("Parent: " + msg);
}
```

Polimorfismo (Polymorphism):

- Ocurre cuando una clase hija (Child) redefine (override) un método de la clase padre (Parent) y el método correspondiente se llama según el tipo del objeto en tiempo de ejecución.
- En tu código, la clase Child redefine el método display que toma un int como parámetro:

```
class Child extends Parent {
    void display(int num) {
        System.out.println("Child: " + num);
    }
}
```

En el main, se crea una instancia de Child referenciada como Parent, y cuando se llama a display(5), se ejecuta la versión del método en Child debido al polimorfismo.

****

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?
El método join se utiliza para esperar a que ambos hilos terminen su ejecución antes de imprimir el valor de count. Dado que cada hilo incrementa count 1000 veces y hay dos hilos, el valor final de count será 2000.

****

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?

Debido a que `Dog` y `Cat` sobrescriben el método `makeSound` de `Animal`, se llamará a la implementación correspondiente de cada clase. Por lo tanto, la salida será la opción 2.

Nombre: Gina Belén Molina Gutiérrez
Semana: 3

****

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?

- En el método `main`, se crea un objeto `Derived` pero se referencia como tipo `Base`.
- Cuando se llama a `obj.show()`, se ejecuta el método `show` de `Derived` debido a la naturaleza polimórfica de Java.
- `FileNotFoundException` es una subclase de `IOException`, por lo que el código está bien desde el punto de vista de la firma del método.

****

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?

Dado que un hilo está incrementando `count` 1000 veces y el otro hilo está decrementando `count` 1000 veces, el valor final de `count` después de que ambos hilos terminen será 0. Por lo tanto, la salida 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?
Dado que el tipo genérico de `Box` es `String` y el valor que se establece es `"Hello"`, el método `getItem` detectará que `item` es una instancia de `String`, realizará el cast inseguro y devolverá el valor sin lanzar una excepción. El valor impreso será `"Hello"`. Por lo tanto, la salida será `Hello`.

Nombre: Gina Belén Molina Gutiérrez

****

Ejercicio 11

Considera el siguiente bloque de código:

```
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");
   }
}
```

Resultado:

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

Esto lanza una excepción `java.lang.ClassCastException` en tiempo de ejecución cuando se intenta ejecutar `objetoHija3.llamarClase()`.

****

Ejercicio 12

Considera el siguiente bloque de 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 sería la salida en consola al ejecutar este código?

1) Animal sound Wau Wau compilation error

2) Compilation Error

3) Animal sound Wau Wau Animal sound

4) Animal sound

¿Cuál crees que es la respuesta correcta?
La conversión inválida `Dog tres = (Dog) new Animal();` no compila, por lo que se produce un error de compilación. Por lo tanto, la respuesta correcta es que el código no compilará debido a la conversión incorrecta.

Ejercicio 13

Considera el siguiente bloque de 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 sería la salida en consola al ejecutar este código?

1) Compilation error

2) Cambios?: 1,hola,4,

3) Cambios?: 1,hola,4,hola2

4) Cambios?: 5,cambiando string,9,

¿Cuál crees que es la respuesta correcta?

- La variable `x` sigue siendo `1`, ya que no se modificó fuera del método.

- La variable `hola` sigue siendo `"hola"`, ya que la modificación dentro del método no afectó el valor original.
- La variable `x2` sigue siendo `4`, ya que la modificación dentro del método `makeSound(Integer, StringBuilder)` no afecta la variable `x2`.

- La variable `hola2` es un `StringBuilder` que ha sido vaciado, por lo que su valor se imprimirá como una cadena vacía.

Ejercicio 14
Considera el siguiente bloque de 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?

    }
```

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

1) solo m1

2) m1 y m2

3) ninguno

4) error compilación

¿Cuál crees que es la respuesta correcta?

1. **Interfaces y herencia**:

- `i1` tiene el método `m1()`.
- `i2` extiende `i1` e introduce un nuevo método `m2()`. Esto significa que `i2` hereda `m1()` de `i1` y añade `m2()`.

2. **Implementación en la clase `animal`**:
   - La clase `animal` implementa tanto `i1` como `i2`. Por lo tanto, `animal` debe implementar todos los métodos que ambas interfaces declaran.
   - Como `i2` extiende `i1`, la implementación en `animal` debe proporcionar implementaciones para todos los métodos de `i1` (que es `m1()`) y todos los métodos de `i2` (que es `m2()`).

En consecuencia, la clase `animal` debe implementar los métodos `m1()` y `m2()` para cumplir con ambas interfaces. Por lo tanto, la clase `animal` debe implementar **ambos métodos**: `m1()` y `m2()`.