INSTITUTO POLITÉCNICO NACIONAL

"ESCOM" (ESCUELA SUPERIOR DE CÓMPUTO)

DOCENTE: RODRIGUEZ CASTILLO MIGUEL ANGEL

INTEGRANTES:

VÁZQUEZ BLANCAS CÉSAR SAID

MENDOZA SEGURA FERNANDO

SEGUNDO CRUZ DANIEL

U.A.: PARADIGMAS DE PROGRAMACION

GRUPO: 3CM5

PRÁCTICA 5

Practica 5.

Planteamiento del problema

Descripción del problema: Aplicar los conocimientos adquiridos para el desarrollo de programas en Java en cuanto al concepto de Clases e instanciar las mismas, utilizando las buenas prácticas de programación en el diseño, implementación, pruebas y depuración del mismo.

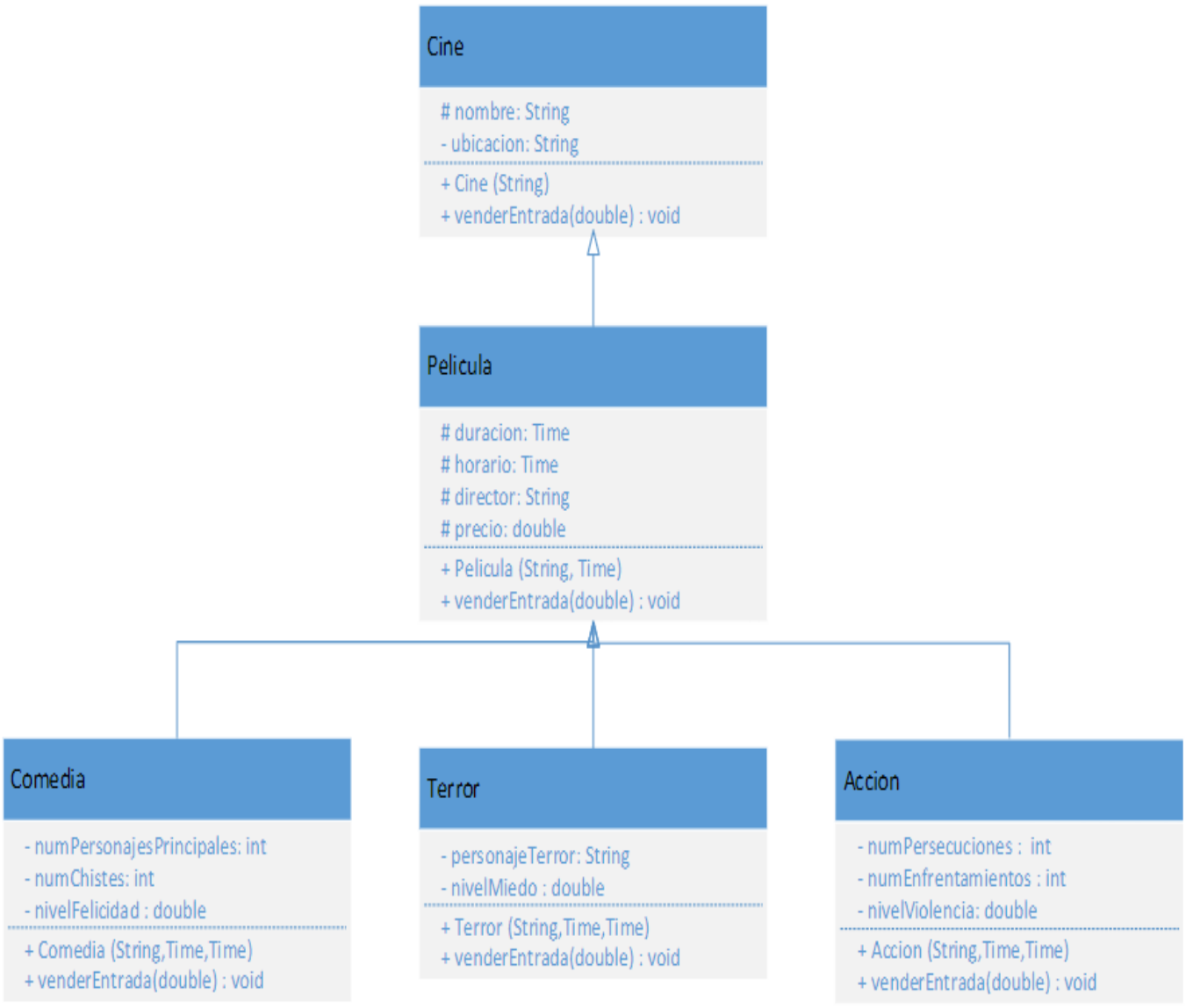
Entrada: Como entrada te datos tenemos todos los atributos dentro de las clases en el diagrama UML, es decir que tendríamos el nombre, ubicación, duración, horario, director, precio, numPersonajesPrincipales, numChistes, nivelFelicidad, personajeTerror, nivelMiedo, numPersecuciones, numEnfrentamientos y nivelViolencia. Las demás líneas de código refieren a los métodos y el desarrollo del programa lo cual entraría en la parte del proceso.

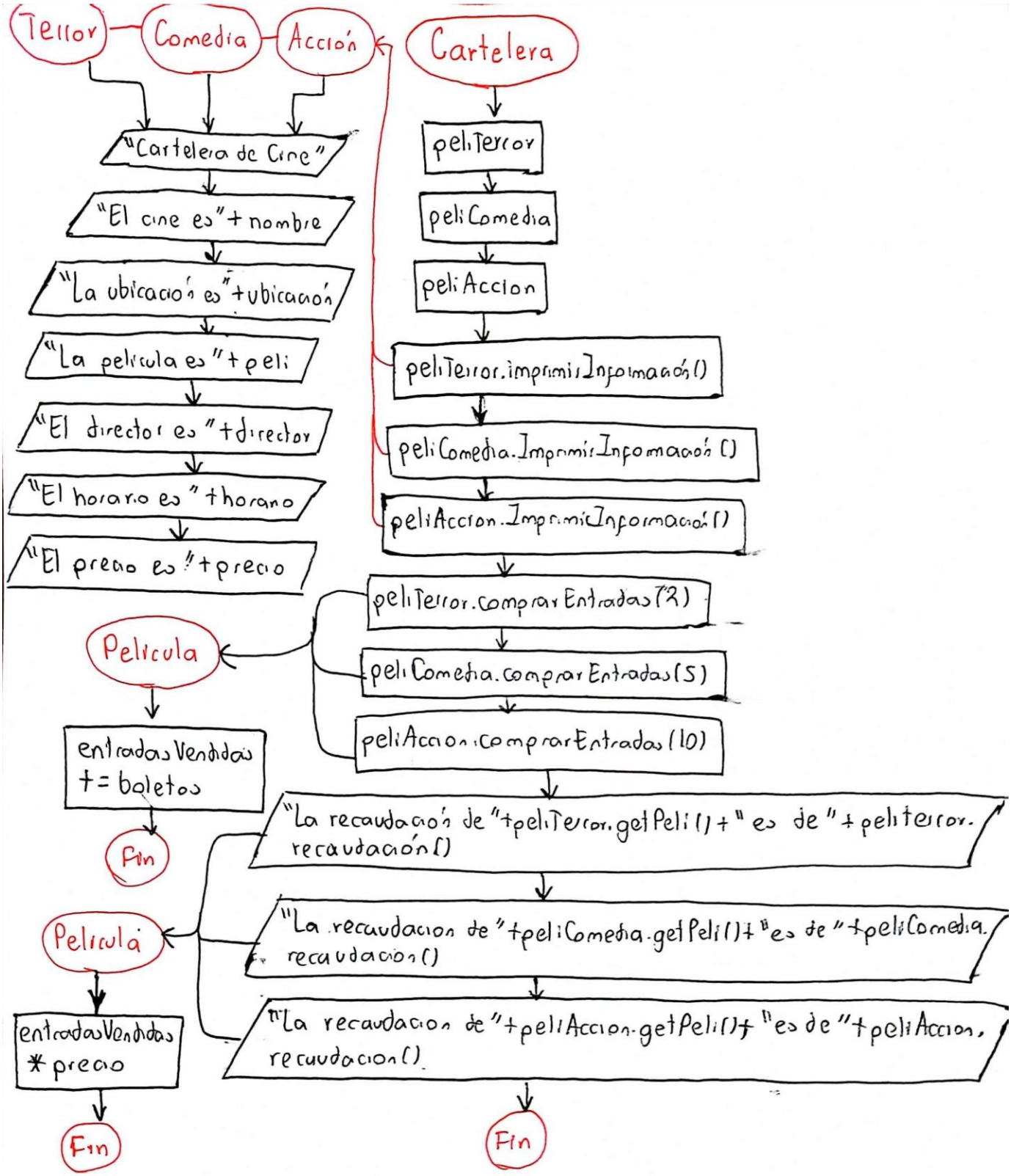
Proceso: En el procesamiento tenemos todo lo que es la implementación de los métodos y el uso de los constructores para el funcionamiento correcto del programa. Al ser una practica referente al uso de la "herencia" debemos analizar múltiples clases ya que debido a la practica tenemos 5 clases diferentes, dentro de las cuales tenemos una superclase o clase base que hereda sus funciones y atributos a 4 subclases o clases derivadas. La primera clase padre tiene una clase hija que a su vez es clase padre de otras tres clases hijas. En esencia todos los métodos creados en base al diagrama UML junto con los métodos set y get, sirven para que el programa imprima una cartelera de cine con al menos una película de cada género (terror, comedia y acción) mostrando los siguientes datos: nombre del cine, ubicación del cine y de cada película su nombre, director, horario, precio de entrada, considerando que dependiendo del genero que sea, se imprimirán los atributos de la clase correspondiente a dicho género. Finalmente, también se implementan métodos para la compra de entradas y mediante estos métodos compraríamos dos entradas para una película de terror, cinco entradas para un de comedia y 10 para una película de acción.

Salida: En cuanto a salida solamente tenemos los datos en impresión de la pantalla comentados en el proceso, es decir, que en la consola se mostraran los precios, nombres de las películas, lo recaudado por cada cine o película según el numero de entradas que compramos, etcétera.

DIAGRAMA UML

Unified Modeling Language




Terror
Comedia
Acción
Cartelera
"Cartelera de Cine"
"El cine es" + nombre
"La ubicación es" + ubicación
"La pelicula es" + peli
"El director es" + director
"El horario es" + horario
"El precio es" + precio
peliTerror
peliComedia
peliAccion
peliTerror.imprimirInformación()
peliComedia.ImprimirInformación()
peliAccion.ImprimirInformación()
peliTerror.comprarEntradas(2)
peliComedia.comprarEntradas(5)
peliAccion.comprarEntradas(10)
Pelicula
entradasVendidas += boletos
Fin
"La recaudación de" + peliTerror.getPeli() + "es de" + peliTerror.recaudación()
Pelicula
"La recaudacion de" + peliComedia.getPeli() + "es de" + peliComedia.recaudacion()
entradasVendidas * precio
Fin
"La recaudacion de" + peliAccion.getPeli() + "es de" + peliAccion.recaudacion()
Fin

Implementación de la solución

Código desarrollado en lenguaje de programación

CLASE “Cine”

En este apartado analizaremos el código y lo explicaremos conforme vamos profundizando en el mismo para que de igual manera se valla entendiendo poco a poco.

Cine.java > {} Practica5

```java
package Practica5;

public class Cine {
    protected String nombre;
    protected String ubicacion;

    public Cine(String nombre, String ubicacion){
        this.nombre=nombre;
        this.ubicacion=ubicacion;
    }
    public String getNombre() {
        return nombre;
    }
    public String getUbicacion() {
        return ubicacion;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public void setUbicacion(String ubicacion) {
        this.ubicacion = ubicacion;
    }
    public void venderEntrada(double precioEntrada){
        System.out.println(x:"Entrada Vendida con exito");
    }
```

Como podemos notar, primero definimos todo dentro de un package “practica 5” en el cual estarán dentro todas las clases del programa. Después se define la clase cine la cual contiene dos atributos (nombre y ubicación) del tipo protected para que las clases hijas y la misma clase cine puedan acceder a los atributos.

Una vez hechos los atributos, comenzamos a definir el constructor de Cine junto con los métodos getters y setters para obtener y asignar valores a los atributos de forma controlada así ninguna clase puede modificar desde fuera de los mismos los valores de dichos atributos, es un principio de encapsulación, al mismo tiempo, puedes codificar ciertas condiciones o realizar cambios en los datos de entrada antes de pasarlos al atributo de la clase.

El constructor se utiliza para definir que sucede en el momento que creamos una instancia de una clase mediante el operador new,el constructor nos permite asignar valores por defecto a los campos o llamar a otro constructor o método si así se desea.

Como ultimo método tenemos “public void venderEntrada(double precioEntrada)” el cual recibe como parámetro un dato de tipo double que será el precio de una entrada de cine, el método también solo imprime en pantalla el mensaje “Entrada Vendida con éxito” mediante un System.out.println().

Antes de comenzar a analizar la clase, debemos dejar en claro que para poder utilizar la herencia existe la palabra clave “extends” la cual se puede usar para crear una subclase a partir de clases personalizadas, así como sus objetos incorporados.

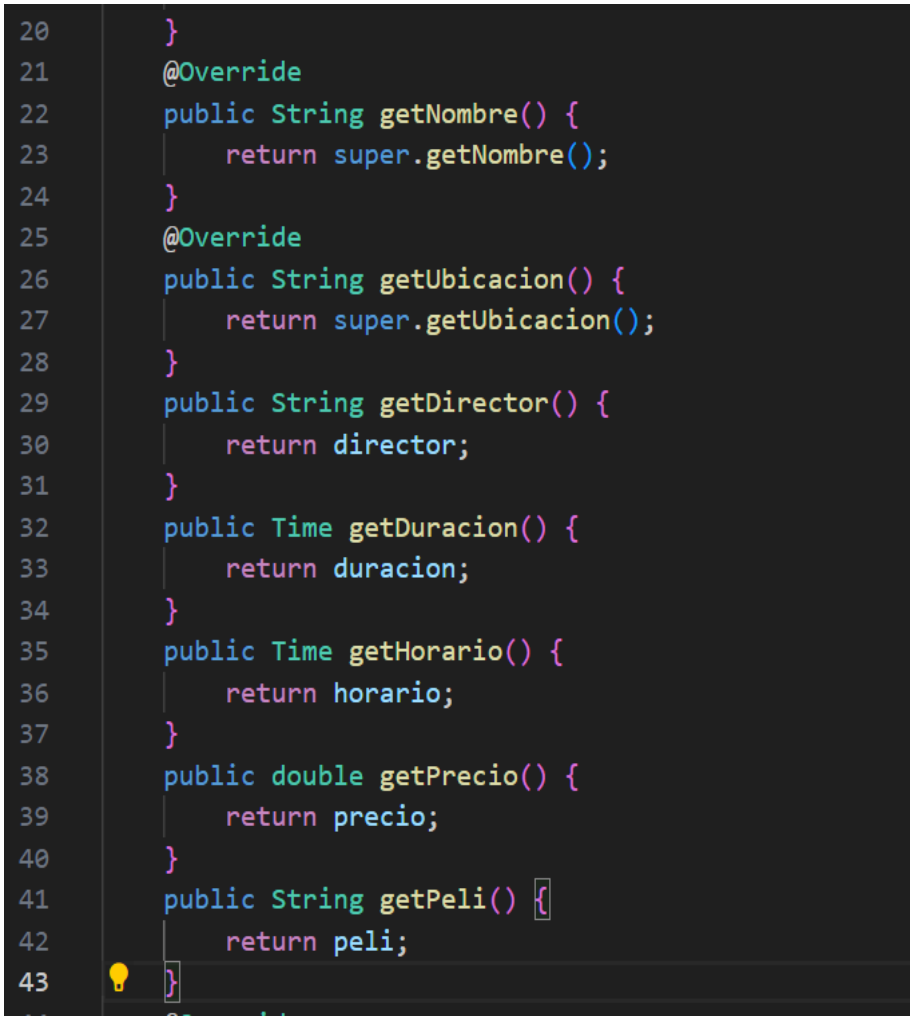
Pelicula.java > Pelicula > peli

```java
package Practica5;
import java.sql.Time;

public class Pelicula extends Cine{
    protected Time duracion;
    protected Time horario;
    protected String director;
    protected double precio;
    protected String peli;
    protected int entradasVendidas;

    public Pelicula(Time duracion, Time horario,String director, double precio,String nombre, String ubicacion,String
        super(nombre,ubicacion);
        this.duracion=duracion;
        this.horario=horario;
        this.director=director;
        this.precio=precio;
        this.peli=peli;
        this.entradasVendidas=0;
    }
    @Override
    public String getNombre() {
        return super.getNombre();
    }
```

```java
Pelicula(Time duracion, Time horario,String director, double precio,String nombre, String ubicacion,String peli){
er(nombre,ubicacion);
s.duracion=duracion;
s.horario=horario;
s.director=director;
s.precio=precio;
s.peli=peli;
s.entradasVendidas=0;
```

Primero en el código podemos notar que al definir la clase Pelicula se utiliza el ya comentando extends lo que indica que es una clase heredada de la clase Cine por lo que obtiene todos sus atributos y métodos para poder utilizarlos. De igual forma ya en la definición de la clase se establecen los atributos y sus tipos de datos junto con el constructor de “Pelicula” el cual recibe todos los atributos tanto los propios de la clase Pelicula como los de la clase Cine. Es necesario aclarar que ya no es necesario inicializar los atributos de la clase cine pues ya se inicializaron por lo que podemos utilizar la palabra clave super la cual es usada para acceder y llamar funciones o atributos de la superclase o clase padre de un objeto.

Los atributos que no son heredados se deben inicializar con los this.nombre = nombre, ya que son propios de la clase Pelicula.



```
20      }
21      @Override
22      public String getNombre() {
23          return super.getNombre();
24      }
25      @Override
26      public String getUbicacion() {
27          return super.getUbicacion();
28      }
29      public String getDirector() {
30          return director;
31      }
32      public Time getDuracion() {
33          return duracion;
34      }
35      public Time getHorario() {
36          return horario;
37      }
38      public double getPrecio() {
39          return precio;
40      }
41      public String getPeli() {
42          return peli;
43      }
```

Después se definen los métodos get y set, lo único que cambia es que ahora se anexa el @Override. La anotación @Override simplemente se utiliza, para forzar al compilador a comprobar en tiempo de compilación que estás sobrescribiendo correctamente un método, y de este modo evitar errores en tiempo de ejecución, los cuales serían mucho más difíciles de detectar. La Sobreescritura es la forma por la cual una clase que hereda puede re-definir los métodos de su clase Padre, de esta manera puede crear nuevos métodos con el mismo nombre de su superclase. De igual manera se utiliza la palabra reservada super para acceder a los métodos de la clase Cine.

```java
@Override
public void setNombre(String nombre) {
    super.setNombre(nombre);
}
@Override
public void setUbicacion(String ubicacion) {
    super.setUbicacion(ubicacion);
}
public void setDirector(String director) {
    this.director = director;
}
public void setDuracion(Time duracion) {
    this.duracion = duracion;
}
public void setHorario(Time horario) {
    this.horario = horario;
}
public void setPrecio(double precio) {
    this.precio = precio;
}
public void setPeli(String peli) {
    this.peli = peli;
}
public int getEntradasVendidas() {
    return entradasVendidas;
}
```

```java
}
public void setEntradasVendidas(int entradasVendidas) {
    this.entradasVendidas = entradasVendidas;
}
```

Ahora pasaremos a la definición de los métodos que son parte del proceso de la venta y compra de entradas.

```
    public void venderEntrada(double precioEntrada){
        if(precioEntrada==precio){
        System.out.println("Entrada vendida en :"+precioEntrada);
        }else if(precioEntrada<precio){
        System.out.println(x:"Cantidad insuficiente de dinero");
        }else{
            System.out.println("Su cambio es "+(precioEntrada-precio));
        }
    }
    public void comprarEntradas(int boletos){
        entradasVendidas+=boletos;
    }
    public double recaudacion(){
        return entradasVendidas*precio;
    }
}
```

En el método venderEntrada(double precioEntrada) se recibe como parámetro un dato de tipo double el cual es el precio de la entrada en el cine una vez recibido se ejecuta el condicional if el cual pregunta si el precioEntrada es igual al precio del boleto definido en la declaración de los objetos, de ser así se devuelve el mensaje del primer System.out.println, en caso de que el precioEntrada fuera menor que precio se mostraría el mensaje de “Cantidad insuficiente de dinero”. El último caso condicional else era en caso de que precioEntrada no fuera igual a precio y tampoco fuera menor, es decir, se definiera una cantidad mayor , se mostraría el mensaje de “Su cambio es”+(precioEntrada-precio).}

El método de comprarEntradas(int boletos) la cual recibe como parámetro un int boletos, numero entero de boletos y ya dentro de la función tenemos un incremento en la variable de entradasVendidas la cual empezó en cero, pero se le suman los boletos de la función. La función recaudación solo retorna el valor de una multiplicación entre las entradas vendidas por el precio de estas.

Cabe mencionar que se añade el import java.sql.Time; para poder utilizar los datos de tipo Time. El tipo de datos time solo almacena valores de hora basados en un reloj de 24 horas. El tipo de datos time tiene un intervalo de 00:00:00,0000000 a 23:59:59,9999999 con una precisión de 100 nanosegundos. El valor predeterminado es 00:00:00,0000000 (medianoche).

CLASE "Comedia"

```java
package Practica5;
import java.sql.Time;
public class Comedia extends Pelicula{
    private int numPP;
    private int numChistes;
    private double nivelFelicidad;

    public Comedia(String nombre,String ubicacion,String peli,Time duracion,Time horario,String director,double preci
        super(duracion, horario, director, precio, nombre, ubicacion,peli);
        this.numPP=numPP;
        this.numChistes=numChistes;
        this.nivelFelicidad=nivelFelicidad;
    }
    @Override
    public String getDirector() {
        return super.getDirector();
    }
    @Override
    public Time getDuracion() {
        return super.getDuracion();
    }
```

```java
peli,Time duracion,Time horario,String director,double precio,int numPP,int numChistes,double nivelFelicidad){
 ubicacion,peli);
```

En esta clase se siguen los mismos principios de las anteriores, ,es decir, la herencia mediante el extends para indicar que Comedia es clase hija de la clase Pelicula, la definición de su constructor con el uso de super para la herencia de atributos e inicialización de los atributos propios de la clase, los métodos heredados colocando @Override junto con los métodos getters y setters propios de la clase.

```java
    @Override
    public Time getHorario() {
        return super.getHorario();
    }
    @Override
    public String getNombre() {
        return super.getNombre();
    }
    @Override
    public double getPrecio() {
        return super.getPrecio();
    }
    @Override
    public String getUbicacion() {
        return super.getUbicacion();
    }
    @Override
    public String getPeli() {
        return super.getPeli();
    }
```

```java
    public double getNivelFelicidad() {
        return nivelFelicidad;
    }
    public int getNumChistes() {
        return numChistes;
    }
    public int getNumPP() {
        return numPP;
    }
    @Override
    public void setDirector(String director) {
        super.setDirector(director);
    }
    @Override
    public void setDuracion(Time duracion) {
        super.setDuracion(duracion);
    }
    @Override
    public void setHorario(Time horario) {
        super.setHorario(horario);
    }
```

```java
@Override
public void setNombre(String nombre) {
    super.setNombre(nombre);
}
@Override
public void setPrecio(double precio) {
    super.setPrecio(precio);
}
@Override
public void setUbicacion(String ubicacion) {
    super.setUbicacion(ubicacion);
}
@Override
public void setPeli(String peli) {
    super.setPeli(peli);
}
public void setNivelFelicidad(double nivelFelicidad) {
    this.nivelFelicidad = nivelFelicidad;
}
```

```java
public void setNumChistes(int numChistes) {
    this.numChistes = numChistes;
}
public void setNumPP(int numPP) {
    this.numPP = numPP;
}
```

```java
@Override
public void venderEntrada(double precioEntrada){
    if(precioEntrada==precio){
    System.out.println("Entrada de "+nombre+" comedia vendida en :"+precioEntrada);
    }else if(precioEntrada<precio){
    System.out.println(x:"Cantidad insuficiente de dinero");
    }else{
        System.out.println("Su cambio es "+(precioEntrada-precio));
    }
}
```

En el metodo de venderEntrada(double precioEntrada) en la clase comedia solo cambio un poco en cuanto a la muestra de los mensajes pero los casos siguen siendo los mismso.

```java
    public void imprimirInformacion(){
        System.out.println(x:"\t\tCARTELERA DE CINE");
        System.out.println(x:"\t\tPELICULA DE COMEDIA");
        System.out.println("El Cine es: "+nombre);
        System.out.println("La Ubicacion es: "+ubicacion);
        System.out.println("La pelicula es: "+peli);
        System.out.println("El director es: "+director);
        System.out.println("El horario es: "+horario);
        System.out.println("El precio es: "+precio);
        System.out.println("El numero de personajes es: "+numPP);
        System.out.println("El numero de chistes es: "+numChistes);
        System.out.println("El nivel de felicidad es:"+nivelFelicidad);
    }
}
```

Finalmente tenemos la funcion imprimirInformacion() pero solamente son una serie de System.out.println los cuales muestran en pantalla todos los datos definidos en los objetos de cada pelicula y atributos que tienen.

A partir de aquí las clases Comedia, Terror y Accion son clases hijas de Pelicula y son muy similares en cuanto a metodos pues heredan la mayoria, siendo el mismo caso con los atributos pero en algunos casos agregando sus propios atributos como nivel de miedo, felicidad o numero de persecuciones y añaden sus respectivos getters y setters.

CLASE "Terror"

```java
package Practica5;

import java.sql.Time;

public class Terror extends Pelicula {
    private String PersonajeTerror;
    private double nivelTerror;

    public Terror(String nombre,String peli,String ubicacion,Time duracion,Time horario,String director,double precio
        super(duracion, horario, director, precio, nombre, ubicacion,peli);
        this.PersonajeTerror=PersonajeTerror;
        this.nivelTerror=nivelTerror;

    }
    @Override
    public String getDirector() {
        return super.getDirector();
    }
    @Override
    public Time getDuracion() {
        return super.getDuracion();
    }
    @Override
    public Time getHorario() {
        return super.getHorario();
```

```java
ng ubicacion,Time duracion,Time horario,String director,double precio,String PersonajeTerror,double nivelTerror){
o, nombre, ubicacion,peli);
```

```java
@Override
public String getNombre() {
    return super.getNombre();
}
@Override
public double getPrecio() {
    return super.getPrecio();
}
@Override
public String getUbicacion() {
    return super.getUbicacion();
}
@Override
public String getPeli() {
    return super.getPeli();
}
public double getNivelTerror() {
    return nivelTerror;
}
public String getPersonajeTerror() {
    return PersonajeTerror;
}
```

```java
@Override
public void setDirector(String director) {
    super.setDirector(director);
}
@Override
public void setDuracion(Time duracion) {
    super.setDuracion(duracion);
}
@Override
public void setHorario(Time horario) {
    super.setHorario(horario);
}
@Override
public void setNombre(String nombre) {
    super.setNombre(nombre);
}
@Override
public void setPrecio(double precio) {
    super.setPrecio(precio);
}
@Override
public void setUbicacion(String ubicacion)
    super.setUbicacion(ubicacion);
}
```

```java
@Override
public void setPeli(String peli) {
    super.setPeli(peli);
}
public void setNivelTerror(double nivelTerror) {
    this.nivelTerror = nivelTerror;
}
public void setPersonajeTerror(String personajeTerror) {
    PersonajeTerror = personajeTerror;
}

@Override
public void venderEntrada(double precioEntrada){
    if(precioEntrada==precio){
    System.out.println("Entrada de "+nombre+" vendida en :"+precioEntrada);
    }else if(precioEntrada<precio){
    System.out.println(x:"Cantidad insuficiente de dinero");
    }else{
        System.out.println("Su cambio es "+(precioEntrada-precio));
    }
}
```

```
    public void imprimirInformacion(){
        System.out.println(x:"\t\tCARTELERA DE CINE");
        System.out.println(x:"\t\tPELICULA DE TERROR");
        System.out.println("El Cine es: "+nombre);
        System.out.println("La Ubicacion es: "+ubicacion);
        System.out.println("La pelicula es: "+peli);
        System.out.println("El director es: "+director);
        System.out.println("El horario es: "+horario);
        System.out.println("El precio es: "+precio);
        System.out.println("El numero del personaje de terror es: "+PersonajeTerror);
        System.out.println("El nivel de miedo es: "+nivelTerror);
    }
}
```

CLASE "Accion"

```java
package Practica5;
import java.sql.Time;
public class Accion extends Pelicula{
    private int numPersecuciones;
    private int numEnfrentamientos;
    private double nivelViolencia;

    public Accion(String nombre,String ubicacion,String peli,Time duracion,Time horario,String director,double precio
        super(duracion, horario, director, precio, nombre, ubicacion,peli);
        this.nivelViolencia=nivelViolencia;
        this.numEnfrentamientos=numEnfrentamientos;
        this.numPersecuciones=numPersecuciones;

    }
    @Override
    public String getDirector() {
        return super.getDirector();
    }
    @Override
    public Time getDuracion() {
        return super.getDuracion();
    }
    @Override
    public Time getHorario() {
        return super.getHorario();
    }
```

```java
@Override
public String getNombre() {
    return super.getNombre();
}
@Override
public double getPrecio() {
    return super.getPrecio();
}
@Override
public String getUbicacion() {
    return super.getUbicacion();
}
@Override
public String getPeli() {
    return super.getPeli();
}
public double getNivelViolencia() {
    return nivelViolencia;
}
public int getNumEnfrentamientos() {
    return numEnfrentamientos;
}
public int getNumPersecuciones() {
    return numPersecuciones;
}
```

```java
@Override
public void setDirector(String director) {
    super.setDirector(director);
}
@Override
public void setDuracion(Time duracion) {
    super.setDuracion(duracion);
}
@Override
public void setHorario(Time horario) {
    super.setHorario(horario);
}
@Override
public void setNombre(String nombre) {
    super.setNombre(nombre);
}
@Override
public void setPrecio(double precio) {
    super.setPrecio(precio);
}
@Override
public void setUbicacion(String ubicacion) {
    super.setUbicacion(ubicacion);
}
```

```java
@Override
public void setPeli(String peli) {
    super.setPeli(peli);
}
public void setNivelViolencia(double nivelViolencia) {
    this.nivelViolencia = nivelViolencia;
}
public void setNumEnfrentamientos(int numEnfrentamientos) {
    this.numEnfrentamientos = numEnfrentamientos;
}
public void setNumPersecuciones(int numPersecuciones) {
    this.numPersecuciones = numPersecuciones;
}
public void venderEntrada(double precioEntrada){
    if(precioEntrada==precio){
    System.out.println("Entrada de "+ nombre +" vendida en :"+precioEntrada);
    }else if(precioEntrada<precio){
    System.out.println(x:"Cantidad insuficiente de dinero");
    }else{
        System.out.println("Su cambio es "+(precioEntrada-precio));
    }
}
```

```java
    public void imprimirInformacion(){
        System.out.println(x:"\t\tCARTELERA CINE");
        System.out.println(x:"\t\tPELICULA DE ACCION");
        System.out.println("El Cine es: "+nombre);
        System.out.println("La Ubicacion es: "+ubicacion);
        System.out.println("La pelicula es: "+peli);
        System.out.println("El director es: "+director);
        System.out.println("El horario es: "+horario);
        System.out.println("El precio es: "+precio);
        System.out.println("El numero de persecuciones es: "+numPersecuciones);
        System.out.println("El numero de enfrentamientos es:"+numEnfrentamientos);
        System.out.println("El nivel de felicidad es:"+nivelViolencia);
    }
}
```

CLASE "Cartelera"

```java
package Practica5;
import java.sql.Time;
public class Cartelera {
    Run | Debug
    public static void main(String[] args) {
        Time duracionTerror = Time.valueOf(s:"1:35:48");
        Time horario = Time.valueOf(s:"12:00:00");
        Time duracionComedia = Time.valueOf(s:"1:40:27");
        Time horarioComedia = Time.valueOf(s:"1:50:00");
        Time duracionAccion = Time.valueOf(s:"1:47:55");
        Time horarioAccion = Time.valueOf(s:"3:45:00");
        Terror peliTerror = new Terror(nombre:"Americas", peli:"Viernes 13", ubicacion:"Plaza las Americas",duracionT
        Comedia peliComedia=new Comedia(nombre:"Americas", ubicacion:"Plaza las Americas", peli:"Hangover", duracionC
        Accion peliAccion=new Accion(nombre:"Americas", ubicacion:"Plaza las Americas", peli:"Terminator", duracionAc
        peliTerror.imprimirInformacion();
        peliComedia.imprimirInformacion();
        peliAccion.imprimirInformacion();
        peliTerror.comprarEntradas(boletos:2);
        peliComedia.comprarEntradas(boletos:5);
        peliAccion.comprarEntradas(boletos:10);
        System.out.println("La recaudacion de "+peliTerror.getPeli()+" es de: "+peliTerror.recaudacion());
        System.out.println("La recaudacion de "+peliComedia.getPeli()+" es de: "+peliComedia.recaudacion());
        System.out.println("La recaudacion de "+peliAccion.getPeli()+" es de: "+peliAccion.recaudacion());
```

```
, director:"Sean S. Cunningham", precio:65.7, PersonajeTerror:"Jason", nivelTerror:4.2);
Comedia, director:"Todd Phillips", precio:53.3, numPP:4, numChistes:230, nivelFelicidad:4.7);
cion, director:"James Cameron", precio:42.7, numPersecuciones:10, numEnfrentamientos:3, nivelViolencia:3.8);
```

```
n:"Plaza las Americas",duracionTerror , horario, director:"Sean S. Cunningham", precio:65.7, PersonajeTerror:"Jason",
cas", peli:"Hangover", duracionComedia, horarioComedia, director:"Todd Phillips", precio:53.3, numPP:4, numChistes:23
", peli:"Terminator", duracionAccion, horarioAccion, director:"James Cameron", precio:42.7, numPersecuciones:10, numE
```

Esta es nuestra clase pirncipal y es donde primero definimos los horarios y tiempos de cada pelicula gracias al import.sql.Time; Despues instanciamos nuestros objetos y definimos los valores de cada atributo dentro del objeto para que pueda procesarse el programa con los datos recabados.

Finalmente imprimimos la informacion de una pelicula de cada genero mediante el metodo imprimirInformación y hacemos la compra del numero solicitado de boletos según la practica por cada genero de pelicula utilizando el metodo .comprarEntradas(), para que se puedan imprimir lo recaudado por cada pelicula meidante System.out.printn.

Funcionamiento

Prueba de ejecución, resultados de salida y/o capturas de pantalla.

```
                CARTELERA DE CINE
                PELICULA DE TERROR
El Cine es: Americas
La Ubicacion es: Plaza las Americas
La pelicula es: Viernes 13
El director es: Sean S. Cunningham
El horario es: 12:00:00
El precio es: 65.7
El numero del personaje de terror es: Jason
El nivel de miedo es: 4.2
                CARTELERA DE CINE
                PELICULA DE COMEDIA
El Cine es: Americas
La Ubicacion es: Plaza las Americas
La pelicula es: Hangover
El director es: Todd Phillips
El horario es: 01:50:00
El precio es: 53.3
El numero de personajes es: 4
El numero de chistes es: 230
El nivel de felicidad es:4.7
```

```
                CARTELERA CINE
                PELICULA DE ACCION
El Cine es: Americas
La Ubicacion es: Plaza las Americas
La pelicula es: Terminator
El director es: James Cameron
El horario es: 03:45:00
El precio es: 42.7
El numero de persecuciones es: 10
El numero de enfrentamientos es:3
El nivel de felicidad es:3.8
La recaudacion de Viernes 13 es de: 131.4
La recaudacion de Hangover es de: 266.5
La recaudacion de Terminator es de: 427.0
PS C:\Users\mendo\Downloads\java documents\Practica5>
```

Conclusiones Individuales

Vázquez Blancas Cesar Said

En esta práctica se ve la herencia en POO, que es un concepto muy importante en este mismo paradigma, ya que con este paradigma se ahorra código y se puede relacionar una clase con otra, puede ser útil para hacer un recabado de datos en los cuales puedes omitir cosas de algo especifico porque ya tienes esos datos en algo general, por lo que este concepto ayuda a ir de algo general a algo particular, el hecho de heredar de 3 maneras los métodos es interesante ya que puedes utilizar el método super, que te ayuda a traer el método anterior al heredado y además agregarle cosas de esa clase, otra manera es que cada clase tiene su método con el mismo nombre pero cada uno funciona con las reglas de la clase, esto se le conoce como sobreescritura de métodos, dependiendo del objeto que se construya y se mande a llamar será el método que agarra , el ultimo es el método heredado, que este no se declara, solo se usa, este concepto es útil para relacionar y reutilizar códigos de una clase padre, esto hace que la construcción del constructor sea bastante extensa, pues los atributos heredados serán los que se vayan pasando para que el constructor funcione adecuadamente, por lo que la escritura de este se hace bastante extensa y un tanto confusa, sin embargo es un concepto que si bien llega a ser útil, la verdad es que debe de elegirse muy bien en que situaciones se ocupa, porque hay alternativas como agregación y composición que hace que sea mejor y más lógico el tema de tener un programa funcional y legible, ya que la herencia tiene sus limitaciones y complicaciones a la hora de hacer arreglos por ejemplo y saber cómo manejarlos junto con sus métodos para que imprima el método correcto, sin embargo se ve útil para un registro de una persona y que luego se pase a un campo más particular.

Segundo Cruz Daniel

En el transcurso de mi práctica relacionada con el tema de herencia en programación orientada a objetos, he consolidado mi comprensión de este concepto crucial. La herencia, al permitir la creación de clases secundarias basadas en clases existentes, se reveló como una herramienta poderosa para la reutilización de código y la organización eficiente de funcionalidades.

Durante la implementación práctica, pude aplicar la herencia de manera efectiva al diseñar jerarquías de clases que compartían atributos y métodos comunes. Esta estructura jerárquica no solo facilitó la modularidad del código, sino que también contribuyó a una mayor claridad en la estructura del programa.

Sin embargo, también enfrenté desafíos notables. La gestión de la herencia puede volverse compleja, especialmente al enfrentar múltiples niveles de jerarquía o al lidiar con la necesidad de modificar el comportamiento heredado. La consideración cuidadosa de la relación "es un/a" entre las clases y la anticipación de posibles cambios en los requisitos fueron aspectos cruciales para evitar problemas posteriores en la implementación.

Además, la práctica destacó la importancia de equilibrar la flexibilidad inherente a la herencia con la necesidad de mantener un código claro y mantenible. A veces, la sobreutilización de la herencia puede conducir a una estructura demasiado compleja y difícil de comprender, lo que subraya la importancia de evaluar cuidadosamente si la herencia es la mejor solución en cada contexto.

Mendoza Segura Fernando

La practica se enfoca mucho en la utilidad de la Herencia en la programación orientada a objetos y en esencia puede decirse que una forma de ahorro de código ya que al permitirnos otorgarles a otras clases del programa algunos atributos o métodos que ya se han definido en una clase diferente con anterioridad, podemos hacer uso de la herencia. Cabe mencionar que sus usos son muy prácticos pues permite la aplicación e introducción a nuevos conceptos como lo son la sobreescritura de métodos, la implementación de la palabra reservada super para el acceso a atributos o métodos de otra clase que los hereda, o el uso del @Override para evitar errores al momento de estar implementando una sobreescritura. Cabe mencionar que también se utilizo la palabra reservada extends para precisamente poder indicarle a java el uso de una herencia y a su vez poder ir definiendo las relaciones sobre la misma declaración de cada clase, es decir que en el mismo código al iniciar una clase podemos indicar quien es clase hija o heredada de otra clase. Lo demás fue practica de la inicialización y creación de objetos juntos con sus constructores para su utilización en el programa y su ejecución, sin dejar de lado los métodos getter y setter para obtener o aginar datos a los atributos en nuestras clases.

Bibliografía:

IV – Sobrecarga de métodos y constructores, Ing. Carlos Alberto Román Zamitiz, Facultad de Ingeniería, UNAM, Recuperado el 14 de noviembre del 2023, de

http://profesores.fi-b.unam.mx/carlos/java/java_basico4_6.html#:~:text=La%20sobrecarga%20de%20m%C3%A9todos%20es,cu%C3%A1l%20definici%C3%B3n%20de%20m%C3%A9todo%20ejecutar.

Videotutorial Sobrecarga del constructor: destructor. - C# - LinkedIn, linkedin.com, Recuperado el 14 de noviembre del 2023, de

https://es.linkedin.com/learning/c-sharp-programacion-orientada-a-objetos/sobrecarga-del-constructor-destructor#:~:text=La%20sobrecarga%20en%20los%20constructores,%22Estudiante%22%20que%20recib%C3%ADa%20par%C3%A1metros.

Herencia en Java: definición y ejemplos, María Coppola, HubSpot, Recuperado el 1 de Diciembre del 2023 de,

https://blog.hubspot.es/website/que-es-herencia-java

¿Para que sirve la líne @Override en java?, E. Betanzos, stackoverflow, Recuperado el 1 de Diciembre del 2023, de

https://es.stackoverflow.com/