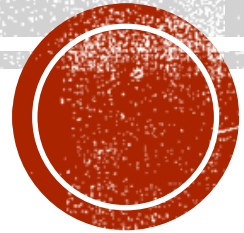
AGENDA DE TEMAS DE HOY

- Ejercicio de Herencia – 2 Niveles
- Proyectos
  - Definición de tema y UML
  - Entregas
  - Fechas
- Examen
  - Contenido
  - Dinámica
- Polimorfismo
  - Conceptos
  - Importancia
  - Implementación

RETO



TEMAS DE PROYECTOS

1. Cajero Automático
2. Tienda en Línea
3. Aerolínea

QUÉ DEBE DE CONTENER

Consideraciones:

- **Se realiza en equipo pero SE SUBE INDIVIDUAL**
- **No hay entregas posterior a la fecha**
- Cada integrante es responsable de su entrega
- TODOS los integrantes deben saber, entender y conocer el funcionamiento al 100% de su proyecto
- Se deberá subir el Código y el UML
- Fecha de entrega, jueves 1 de diciembre 23:59pm
- Recuerda subir 2 veces (para tu nota y para eLumen)

QUÉ DEBE DE CONTENER

Qué debe incluir al menos:

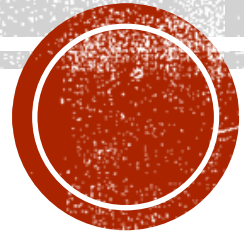
1. 1 Clase padre
2. Al menos 5 clases hijas
3. Herencias entre clases
4. Modificación y accesos de atributos mediante Getters y Setters
5. Cuidar el funcionamiento real del tema (flujo o modularización)

ACTIVIDAD ENTREGABLE

- Reúnete con tu equipo y realiza el diagrama UML del tema que te tocó realizar en tu proyecto/reto
- Basado en esto se evaluará tu trabajo
- Es requisito que tu Profe del el visto bueno ;)
- Tienes 30 minutos para esto :D

A trabajar …

EXAMEN



EXAMEN

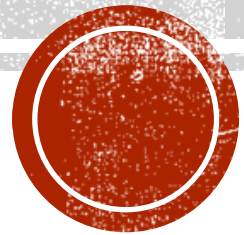
Consideraciones:

- Examen presencial
- Se debe de subir a la eLumen y Canvas
- Duración máxima 100 min
- Llegar puntual.

Estructura:

1. El examen constara de 2 secciones
    1. 30% Parte teórica
    2. 70% Parte Práctica

POLIMORFISMO



POLIMORFISMO

- **Herencia**

La **herencia** ocurre cuando se crea una clase que es hija de otra clase, en ese caso la clase hija hereda todos los atributos y métodos de la clase padre.

- **Polimorfismo**

El **polimorfismo** es cuando una clase hija sobrescribe un método que le fue heredado. Por lo que el polimorfismo permite redefinir el comportamiento de la clase.

Para sobrescribir un método dentro de una clase hija solamente deberemos volver a declarar un método con el mismo nombre que tenía en la clase padre.

POLIMORFISMO

Una **clase puede exhibir comportamientos (métodos) que no tiene definidos en sí misma sino en otras clases que se derivan de ella.** Es decir, una misma clase puede adoptar "muchas formas" diferentes dependiendo de las redefiniciones presentes en clases que se deriven de ella.

Existen dos clases de polimorfismo que se verán a continuación:

- Polimorfismo estático (en tiempo de compilación):
- Polimorfismo dinámico (en tiempo de ejecución):

POLIMORFISMO

Polimorfismo estático (en tiempo de compilación):

- Este ocurre cuando se da un proceso conocido como "early binding" o "static binding" que no es otra cosa distinta a que la memoria que se reserva para un polimorfismo es reservada en tiempo de compilación. Se atribuye este polimorfismo particularmente a la sobrecarga de funciones (métodos.

POLIMORFISMO

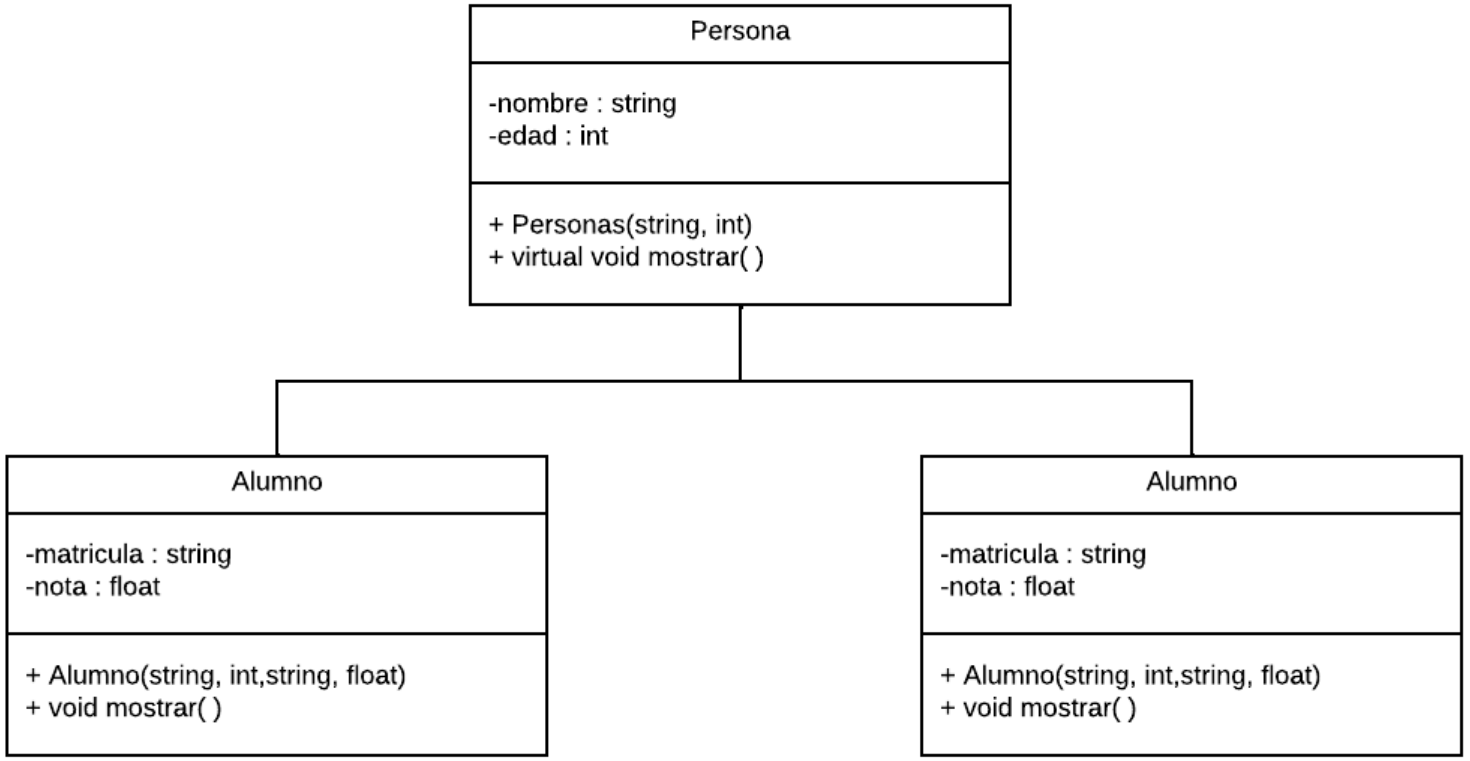
Polimorfismo dinámico (en tiempo de ejecución):

- Se le conoce como dinámico porque sucede en tiempo de ejecución de la aplicación, es decir la memoria que se reserva para este polimorfismo es reservada en el momento que la aplicación lo requiere (proceso conocido como "late binding" o "dynamic binding").
- Esta situación se da mediante el uso de **métodos virtuales** de clases base que son redefinidos en clases derivadas. Es estrictamente necesario el uso de la palabra reservada **`virtual`** ya que este polimorfismo depende de un **proceso en el cual un puntero a funciones consulta una tabla de funciones virtuales para resolver si un método que se invoca es el que está definido en la clase**

POLIMORFISMO

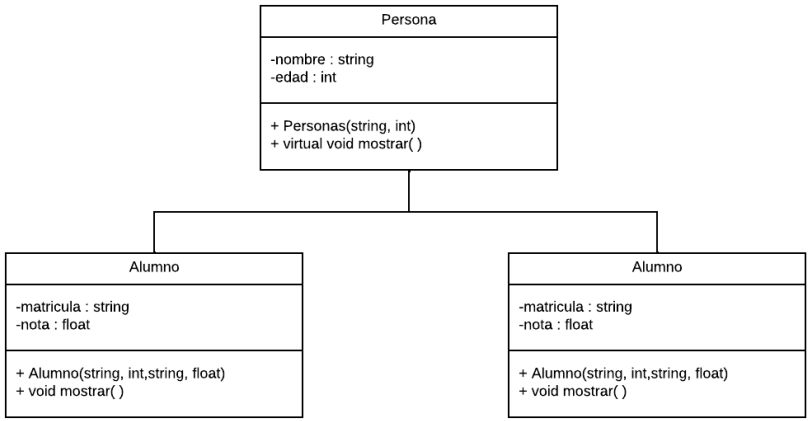
- ¿Dudas?

VEAMOS ESTO A TRAVÉS DE UN EJEMPLO



POLIMORFISMO

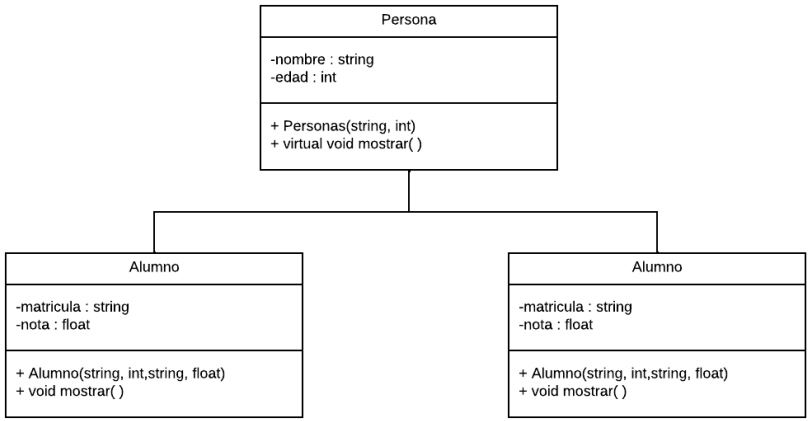
- 1. Definición de clases




```
//Clase Padre o Super Clase
class Persona{
        private:
                string nombre;
                int edad;
        public:
                Persona(string,int);
                virtual void mostrar();
};

//Sub Clase o clase hija
class Alumno : public Persona{
        private:
                string matricula;
                float nota;
        public:
                Alumno(string,int,string,float);
                void mostrar();

};

//Sub Clase o clase hija
class Profesor : public Persona{
        private:
                string nomina;
                float sueldo;
        public:
                Profesor(string,int,string,float);
                void mostrar();
};
```

POLIMORFISMO

- 2. Definición de Constructores



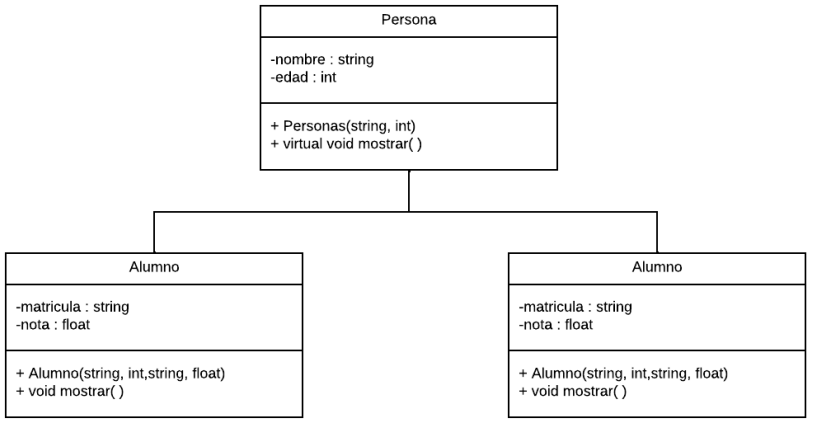

```
//Constructor Padre
Persona::Persona(string _nombre,int _edad){
        nombre=_nombre;
        edad=_edad;
}
```

```
//Constructor hijo
Alumno::Alumno(string _nombre,int _edad, string _matricula, float _nota) :
Persona(_nombre,_edad) {
        matricula=_matricula;
        nota=_nota;

}
```

```
//Constructor hijo
Profesor::Profesor(string _nombre,int _edad, string _nomina, float _sueldo) :
Persona (_nombre,_edad){
        nomina=_nomina;
        sueldo=_sueldo;
}
```

POLIMORFISMO

- 3. Creación de los métodos



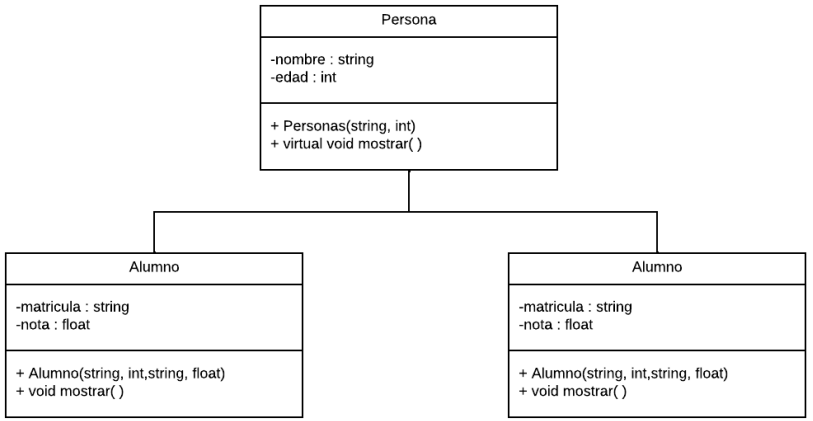

```
void Persona::mostrar(){
        cout<<"\n";
        cout<<"Nombre: "<<nombre<<endl;
        cout<<"Edad: "<<edad<<endl;
}
```

```
void Alumno::mostrar(){
        Persona::mostrar();
        cout<<"Matricula: "<<matricula<<endl;
        cout<<"Nota Final: "<<nota<<endl;

}
```

```
void Profesor::mostrar(){
        Persona::mostrar();
        cout<<"Nomina: "<<nomina<<endl;
        cout<<"Sueldo: $"<<sueldo<<endl;
}
```

POLIMORFISMO

- 4. Método main ( )




```
int main( ){
        Persona *personas[3];

        personas[0]= new Alumno("Michael",19,"A00332211",94.34);
        personas[0]->mostrar();

        personas[1]= new Profesor("Diego",15,"L09999",100000);
        personas[1]->mostrar();

        personas[2]= new Persona("Ana",22);
        personas[2]->mostrar();


        return 0;
}
```

POLIMORFISMO

El polimorfismo promueve la extensibilidad: el software que está escrito para llamar al comportamiento polimórfico se escribe en forma independiente de los tipos de objetos a los cuales se envían los mensajes.

Por lo tanto los nuevos tipos de objetos que pueden responder a los mensajes existentes se pueden agregar a un sistemas, sin modificar el sistema base.

POLIMORFISMO

Ejercicio para la sesión de hoy

Implementar en C++ un script que contenga lo siguiente:

- La clase padre se llamará Seres
- Clases hijas: Humanos y Animales.
- Implementar polimorfismo con el método comer( )