Desarrollo de Sistemas Orientados a Objetos

Profesora Silvia Cañizares – IFTS 29 – Año 2024

Metodología de trabajo

Clase semanal

- ★ Miércoles 18:50 hs.
- ★ Explicación de la teoría fundamental que se aborda en el módulo.
- ★ Quedará grabada y disponible en la plataforma para su posterior visualización.

Sección de Noticias de cursada

- ★ Vía de comunicación para informarles cualquier novedad o modificación en la materia.

Foro de consultas y orientaciones

- ★ Allí podrán realizar comentarios y publicar sus dudas ya sean conceptuales o de código referidas al contenido de la cursada.
- ★ Considerarlo un espacio de intercambio y aprendizaje colaborativo.

Oficinas de trabajo

- ★ De uso obligatorio para intercambiar ideas dentro del equipo y realizar el seguimiento de participación. La información será visible para el grupo y para mi como docente.
- ★ Es importante que puedan dejar el link de acceso a cada documento compartido que realicen, dándome permisos de edición (silvia.canizares@bue.edu.ar) que me faciliten la supervisión del trabajo grupal. Recuerden que, para que puedan visualizarse los cambios que realiza cada integrante, deben compartir el archivo con los mails que utilizan.
- ★ Todos los archivos correspondientes (pueden utilizar enlaces) a los TPs obligatorios y al Proyecto Integrador, que sean trabajados por el equipo, deberán quedar disponibles dentro de este espacio para que pueda brindarles un seguimiento más adecuado.
- ★ Pueden utilizar las herramientas que consideren más adecuadas para cada caso (Ej: Documentos compartidos para documentación, Git quizás para el código, etc.)

Consideraciones importantes

Un Proyecto Integrador Obligatorio

- ★ Con una entrega parcial en Semana 9 y una final en Semana 14.
- ★ Con video grupal de coloquio obligatorio. Forma parte (50%) de la nota de cierre.

Dos Trabajos Prácticos Obligatorios

- ★ Entregas en Semana 7 y 12.
- ★ Forman parte de la calificación final de la materia (50%).

Foros y prácticas formativas

- ★ No son obligatorios ni entregables para corrección.
- ★ Dispondrán de ejemplos para realizar autocorrecciones.

- ★ **Los trabajos pueden ser grupales, de hasta 5 integrantes (consulten si tienen dudas).**
- ★ **Es fundamental que lleven la materia al día y realicen prácticas para incorporar el POO.**

IMPORTANTE

- ★ **La nota de cierre para la aprobación de la materia (mínimo 4) la conformaré en base a los trabajos entregados y sin que deban rendir final.**
- ★ **IGUALMENTE DEBERÁN ANOTARSE para diciembre (más cerca de esa fecha) para que pueda pasarles la calificación de DSOO.**
- ★ **Si aprueban esta materia, pero aún no la correlativa, no puedo subirles la nota de DSOO. Quedaría cursada, y recién al aprobar la correlativa, DEBERÁN ANOTARSE para que pueda pasarles la calificación.**
- ★ **Si se les vence la correlativa, también se les vencerá esta materia.**

CORRELATIVAS

Para aprobar		Deberá haber aprobado	
Código	Nombre	Código	Nombre
1.2.1	Desarrollo de Sistemas Orientado a Objetos	1.1.1	Técnicas de Programación
		1.1.2	Administración de Bases de Datos
		1.1.4	Lógica Computacional
1.2.2	Modelado y Diseño de Software	1.1.1	Técnicas de Programación
2.1.1	Desarrollo de Aplicaciones para Dispositivos Móviles	1.2.1	Desarrollo de Sistemas Orientado a Objetos
		1.2.2	Modelado y Diseño de Software
2.1.2	Metodología de Pruebas de Sistemas	1.1.1	Técnicas de Programación
2.1.5	Desarrollo de Sistemas de Información orientados a la gestión y apoyo a las decisiones	1.2.1	Desarrollo de Sistemas Orientado a Objetos
		1.2.2	Modelado y Diseño de Software
		1.2.5	Aproximación al campo profesional del desarrollo de software
2.2.1	Desarrollo de Sistemas Web (Back End)	2.1.1	Desarrollo de aplicaciones para dispositivos móviles
		2.1.2	Metodología de pruebas de sistemas
		2.1.3	Tecnologías de la información y la comunicación
2.2.2	Desarrollo de Sistemas Web (Front End)	1.2.2	Modelado y diseño de software
2.2.4	Desarrollo e implementación de sistemas en la nube	2.1.1	Desarrollo de aplicaciones para dispositivos móviles
		2.1.3	Tecnologías de la información y la comunicación
		2.1.5	Desarrollo de sistemas de información orientados a la gestión y apoyo a las decisiones
3.1.1	Programación sobre Redes	2.2.1	Desarrollo de Sistemas Web (Back End)
		2.2.3	Ingeniería de Software
3.1.2	Seminario de Profundización y/o Actualización	2.2.3	Ingeniería de Software
3.1.3	Gestión de Proyectos	2.2.3	Ingeniería de Software
3.1.5	Proyecto Integrador	2.2.4	Desarrollo e implementación de sistemas en la nube

Repasando...

La **herencia** permite crear jerarquías de clases, las clases derivadas heredan características y comportamientos de una clase base. En C# tienen sobreescritura de métodos y constructores en las clases derivadas.

La **sobreescritura de métodos** permite redefinir uno heredado, brindando flexibilidad para personalizar comportamientos y adaptarlos a necesidades específicas de la clase derivada.

El **manejo de constructores** en la herencia nos asegura una correcta inicialización de los objetos en una jerarquía de clases. Al crear una instancia de una clase derivada, primero se llama al constructor de la clase base para realizar la inicialización necesaria en la clase base. Luego, se ejecuta el constructor de la clase derivada, donde podemos proporcionar una lógica adicional y personalizada.

Constructores en clases derivadas

Permiten establecer los valores iniciales de los atributos de las clases derivadas, así como invocar los constructores de la clase base para inicializar los atributos heredados, permite una inicialización coherente y controlada de los objetos en una jerarquía de clases, asegurando que los atributos tanto de la clase base como de las clases derivadas estén correctamente establecidos.

Cuando una clase derivada se crea, **primero se llama al constructor de la clase base antes de ejecutar su propio constructor**. Esto asegura que se realice la inicialización necesaria en la clase base antes de continuar con cualquier operación adicional en la clase derivada.

Para manejar los constructores en la herencia en C#, se utiliza la palabra clave **base** para llamar al constructor de la clase base. La llamada se realiza en el bloque de código del constructor de la clase derivada, **como la primera instrucción que se ejecuta.**

La clase base `Animal` con un constructor recibe un parámetro `nombre`. En la clase derivada `Perro`, utilizamos: `base(nombre)` para llamar al constructor de la clase base, pasando el parámetro `nombre` correspondiente.

Al crear una instancia de `Perro` en el método `Main`, se llamará primero al constructor de la clase base `Animal` a través de la llamada `: base(nombre)`. Luego, se ejecutará el constructor de la clase derivada `Perro`. Esto asegura que el objeto `Perro` esté correctamente inicializado tanto en la clase base como en la clase derivada.

```
class Animal
{
    private string nombre;
    public Animal(string nombre)
permiten establecer los valores iniciize: large;">    {
        this.nombre = nombre;
        Console.WriteLine("Se creó un animal llamado: " + nombre);
    }
}
class Perro : Animal
{
    public Perro(string nombre) : base(nombre)
    {
        Console.WriteLine("Se creó un perro llamado: " + getNombre();
    }
}
class Program
{
    static void Main(string[] args)
    {
        Perro perro = new Perro("Firulais");
        // Salida:
        // Se creó un animal llamado: Firulais
        // Se creó un perro llamado: Firulais
    }
```

Métodos en las clases derivadas

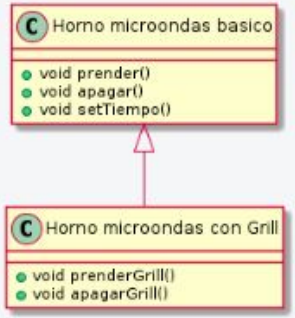
La sobreescritura de métodos es un mecanismo que nos permite redefinir un método heredado de la clase base en una clase derivada. Brinda la flexibilidad de adaptar y modificar el comportamiento según las necesidades específicas de la clase derivada.

Utilizando "override" (como en ToString), indicamos la sobrescritura de un método. Así, podemos implementar lógica adicional, cambiar el comportamiento por completo o extender la implementación original de la clase base.

Este concepto es fundamental para el polimorfismo (4to Pilar), para tratar los objetos de diferentes clases derivadas de manera uniforme a través de una referencia a la clase base.

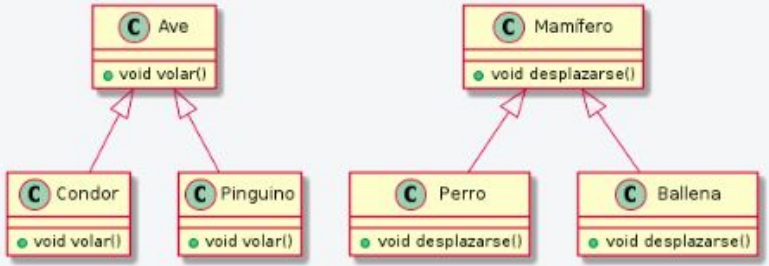
En una clase derivada se pueden añadir nuevos métodos/atributos propios, agregando nuevas características a la nueva clase, como por ejemplo el método ladrar() en una clase perro cuando ésta hereda métodos de Ser Vivo como respirar().

Métodos en las clases derivadas



El modelo con grill de cierta marca es igual al básico, salvo que se le agrega el grill.

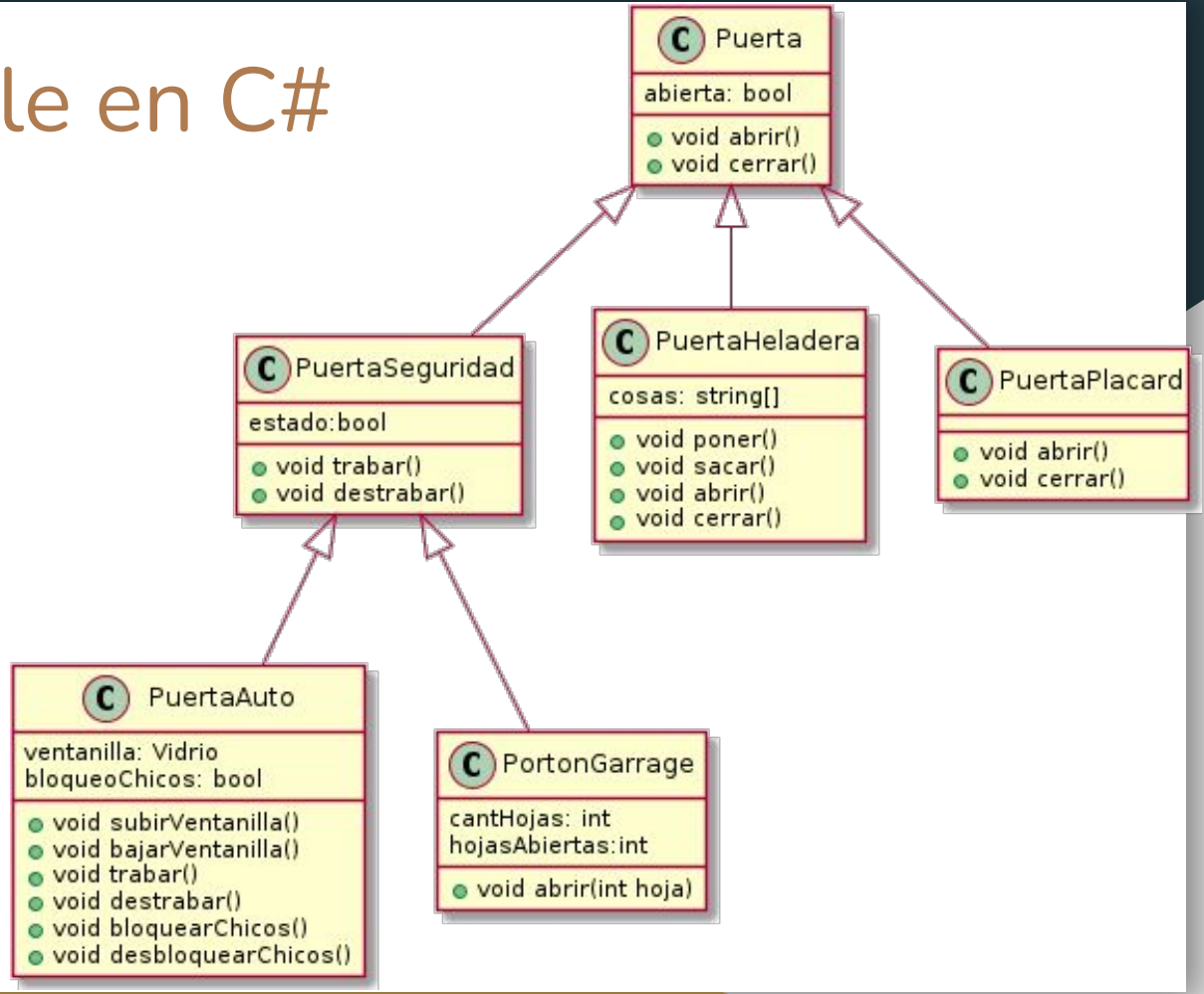
Modificar los métodos heredados de la clase base: el método heredado se redefine completamente, si bien se "llama" igual, su comportamiento puede ser muy distinto.



La capacidad de volar y la forma es muy distinto entre un cóndor (muy alta capacidad) y un pingüino (no vuela) a pesar que ambos son aves. La forma de desplazarse de un perro (camina, corre..) es muy distinta a la de una ballena (nada).

Herencia simple en C#

Puerta solo puede ”abrir” y ”cerrar”. Las otras clases heredan sus propiedades, pero agregándole alguna característica.
PuertaSeguridad representa una puerta que tiene algún tipo de cerradura mientras que PuertaHeladera no tiene ningún tipo de seguridad pero sí puede guardar tener cosas (huevos, botellas de agua, aderezos.) en sus estantes.
PuertaAuto tiene una ventanilla que se puede bajar y subir mientras qué Porton Garage puede tener varias hojas (se puede abrir una hoja para que pase una persona, o 2 para que pase un auto).



En primer lugar se define la clase Puerta; observar que se le agregaron algunos métodos y atributos más que en el diagrama, como un “nombre” para poder diferenciar una puerta de otra puerta.

```
class Puerta {
    bool abierta;
    public void cerrar()  {...}
    public override string ToString() {...}
    public Puerta(string nom) {...}
    protected string getNombre()  {...} }
    string nombre;
    public void abrir()   {...}
```

Para definir la clase hija PuertaSeguridad se escribe el nombre de la clase hija "dos puntos" nombre de la clase padre

```
class PuertaSeguridad : Puerta{
...}
```

Al agregar los dos puntos (:) y el nombre de la clase padre, la nueva clase PuertaSeguridad hereda todos los atributos y métodos de Puerta, además un nuevo atributo “estado” y otros métodos más.

```
class PuertaSeguridad : Puerta {
    bool estado; // true si está trabada, false si no lo esta.
    public PuertaSeguridad(string n) :base(n)  {...}
    public void trabar()  {...}
    public void destrabar()   {...} }
```

Así, en un main se podría declarar y crear un objeto llamado p del tipo Puerta, con los métodos abrir y cerrar, y otro del tipo PuertaSeguridad que puede llamar a “abrir” y “cerrar” definidos en la clase padre Puerta pero también puede ejecutar los nuevos métodos trabar y destrabar.

```
static void Main(string[] args) {
    var p = new Puerta("Comedor");
    p.abrir();
    p.cerrar();
    Console.WriteLine(p);
    var pSeg = new PuertaSeguridad("Tranquera");
    pSeg.abrir();
    pSeg.cerrar();
    pSeg.trabar();
    pSeg.destrabar();
    Console.WriteLine(pSeg); ...}
```

Redefinición de métodos - new

La sobreescritura de métodos permite **redefinir un método que se hereda** para que funcione de acuerdo a nuevas necesidades y no a lo definido en la superclase.

Cuando en un objeto se llama a un método **el compilador comprueba si el método existe en nuestro objeto**, si existe lo usa y si no existe lo busca en la superclase. Esto ocurre así hasta que el compilador encuentra el método definido, busca el método de “abajo hacia arriba”.

En la clase PuertaSeguridad, si la puerta está trabada (tiene un pasador, un cerrojo, llave, etc.) no podría abrirse. Para ello redefinimos el método abrir, para que controle si está trabada o no. *Hay que observar que el nombre del método, los parámetros y lo que devuelve no cambian (sino sería una sobrecarga)*. La clase Puerta define un método abrir() que cambia el valor del atributo abierta y luego es redefinido en la clase PuertaSeguridad con un nuevo método abrir() que antes de abrir la puerta comprueba si está trabada o no.

```cs
using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApp9
{
    3 referencias
    internal class Puerta
    {
        private bool abierta;
        protected string nombre;
        1 referencia
        public Puerta (string nombre)
        {
            this.nombre = nombre;
        }
        1 referencia
        public void abrir()
        {
            abierta = true;
            Console.WriteLine(nombre + ":Abriendo");
        }
        0 referencias
        public virtual void cerrar()
        {
            abierta = false;
            Console.WriteLine(nombre + ":Cerrando");
        }
        1 referencia
        public string getNombre()
        {
            return this.nombre;
        }
    }
}
```

```cs
using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApp9
{
    2 referencias
    internal class PuertaSeguridad : Puerta
    {
        bool estado; // true si está trabada, false si no lo esta.
        1 referencia
        public PuertaSeguridad(string nombre):base(nombre)
        {
            this.estado = true;
        }
        2 referencias
        public new void abrir()
        {
            if (!estado)
                base.abrir(); // abre efectivamente la puerta
            else
                Console.WriteLine(getNombre() + ": No se puede abri
        }
        1 referencia
        public void trabar()
        {
            estado = true;
        }

        1 referencia
        public void destrabar()
        {
            estado = false;
        }
    }
}
```

```cs
using Syste

namespace C
{
    0 referencia
    interna
    {
        0 ref
        sta
        {
        }
    }
}
```

Principal **diferencia entre "new" y "override"**: en cómo se emplean y su implicación en la herencia. "New" se utiliza para ocultar un miembro heredado y proporcionar una nueva implementación independiente, mientras que "override" para reemplazar un método heredado con una nueva implementación que mantiene la relación de herencia. Usar "new" o "override" dependerá de las necesidades, del diseño de clases y relaciones de herencia.

new: crear una nueva implementación del miembro en la clase derivada sin tener en cuenta el heredado de la base (“ocultamiento de miembros”), puede brindar una implementación diferente o adicional para el mismo nombre de miembro en la clase derivada. **OJO: El miembro ocultado de la clase base no es accesible a través de la clase derivada.**

override: indicar que un método de una clase derivada está reemplazando (sobrescribiendo) un método heredado de la base. Se proporciona una implementación del método que tiene el mismo nombre y firma en la clase derivada para que utilice su propia lógica y comportamiento para ese método específico, conservando la relación de herencia y el polimorfismo. Se puede acceder al comportamiento de la clase base mediante la palabra clave "base", lo que te permite extender o modificar la implementación original.

Entorno de datos y codificación

Para trabajar en el entorno de datos y codificar el sistema, comenzamos con la programación y la primer pantalla es la característica ventana que nos pide usuario y contraseña.

Para la validación se busca un registro de usuario y si existe el ingresado, se valida la cadena de caracteres ingresada como contraseña. Ese registro se encuentra almacenado en una estructura que en nuestro caso es una base de datos. El resto de los datos que manipula el sistema también se deben alojar en el mismo entorno para recuperarlos cuando sea necesario. Para lograrlo debemos ver cómo establecemos esa comunicación y cómo codificamos los ingresos y consultas de datos. Para la conexión lo primero que determinamos es el sistema gestor de base de datos a usar, en la materia de base de datos las consultas se hicieron en el entorno de MySQL, por lo tanto el proyecto integrador se conectará con una base de datos de MySQL. En el aula virtual encontrarán el paso a paso dentro del libro: "Entorno de datos y codificación del sistema del Instituto".

A2.Foro. Práctica sobre herencia

Práctica Formativa: Herencia

- **Tipo:** Práctica formativa
- **Objetivo:** Practicar Herencia

Descripción:
Vamos a retomar el enunciado trabajado la semana 8 y le agregaremos funcionalidad de acuerdo a lo pedido.

¿Te acordás del enunciado? *Hacé clic en el botón para leerlo de nuevo.*

Para hacer:

Para cumplir con los requisitos, debemos crear constructores, getters, setters y el método mostrarDatos en ambas clases de productos.

Tendremos una función llamada "calcularPrecioTotal" , que según cada clase hará una cosa u otra, a esta función le pasaremos un número siendo la cantidad de productos

- En Producto, simplemente seria multiplicar el precio por la cantidad de productos especificados.
- En Perecedero, aparte de lo que hace producto, el precio se reducirá dependiendo de los días a caducar según las siguientes reglas:

1. Si le queda 1 día para caducar, se reducirá 4 veces el precio final.
2. Si le quedan 2 días para caducar, se reducirá 3 veces el precio final.
3. Si le quedan 3 días para caducar, se reducirá a la mitad de su precio final.

- En NoPerecedero, el cálculo es igual al de Producto sin ninguna modificación adicional.

Para probar nuestro sistema, crearemos una clase de prueba llamada "Test" y crearemos una lista de productos.

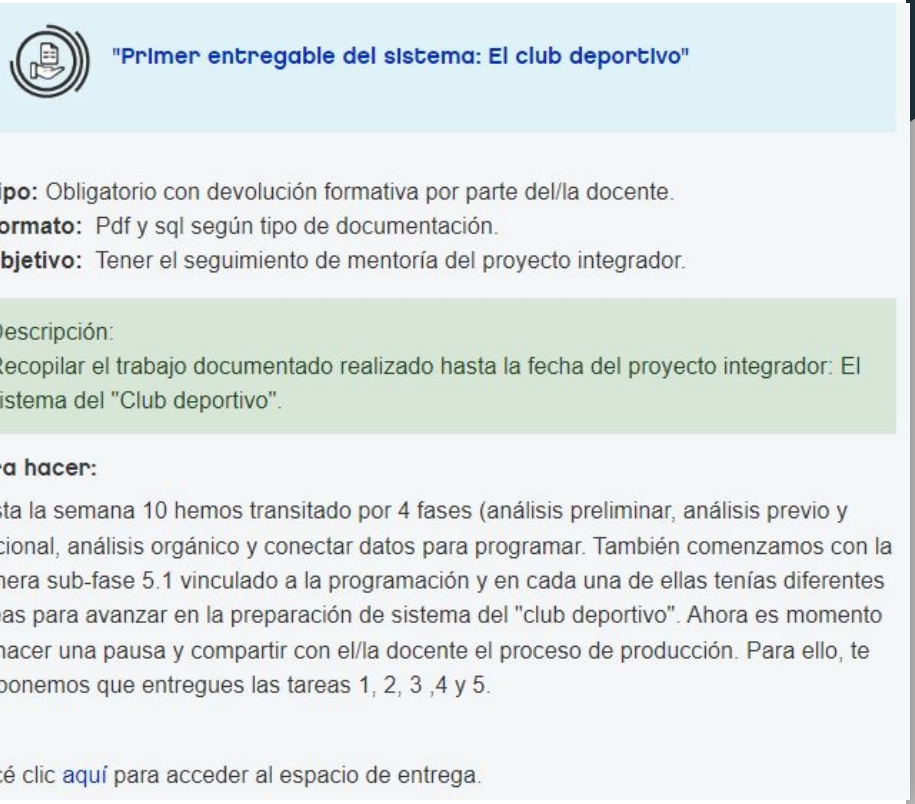
Luego, mostraremos el precio total de vender 5 productos de cada tipo.

Podés crear tus propios objetos de producto dentro del array para realizar la prueba.

El club deportivo (Fase 4) - Conectar datos para programar y (Fase 5) Programar el menú principal y los nuevos socios

Primer entregable

Material sobre el Instituto - EN DRIVE



"Primer entregable del sistema: El club deportivo"

- **Tipo:** Obligatorio con devolución formativa por parte del/la docente.
- **Formato:** Pdf y sql según tipo de documentación.
- **Objetivo:** Tener el seguimiento de mentoría del proyecto integrador.

Descripción:
Recopilar el trabajo documentado realizado hasta la fecha del proyecto integrador: El sistema del "Club deportivo".

Para hacer:

Hasta la semana 10 hemos transitado por 4 fases (análisis preliminar, análisis previo y funcional, análisis orgánico y conectar datos para programar. También comenzamos con la primera sub-fase 5.1 vinculado a la programación y en cada una de ellas tenías diferentes tareas para avanzar en la preparación de sistema del "club deportivo". Ahora es momento de hacer una pausa y compartir con el/la docente el proceso de producción. Para ello, te proponemos que entregues las tareas 1, 2, 3 ,4 y 5.

Hacé clic aquí para acceder al espacio de entrega.