**Clase 3.1**

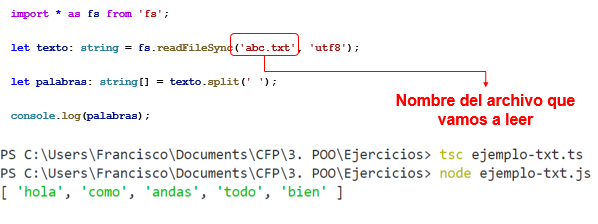
* Crear una cuenta en Github
* Clonarse el repositorio • git clone URL-REPOSITORIO
* Agregar un archivo a la carpeta del repo
* Subirlo al repo

**Clase 3.1.1**

* Demostración en Clase
  + Crear un proyecto
  + Implementar una determinada funcionalidad
  + Correr el archivo con node
  + Definir una tarea en package.json
  + Correr el archivo con NPM
* Ejercicios - En Clase
  + Crear un proyecto
  + Crear dos archivos con funcionalidades diferentes
  + Definir las tareas asociadas y ejecutarlas
* Ejercicios - Fuera de Clase
  + Crear Proyecto NPM
  + Buscar en la página alguna dependencia e instalarla (además de readline-sync)
  + Desarrollar alguna funcionalidad que utilice la dependencia instalada
  + Crear dos archivos con funcionalidades diferentes
    - Cada uno utilizando una librería diferente, sin contar readline-sync
  + Definir las tareas asociadas y ejecutarlas
  + Subir proyecto a GitHub y pasar link por Slack

**Clase 3.1.2**

* Demostración Ejemplo Básico de Tipos
  + Implementar una función que sume los elementos de un arreglo
  + Forzar algún tipo de error a modo de ejemplificación
* Ejercicios - En Clase
  + Ejercicio 1 (Partir de los ejercicios de NPM)
    - Adaptar la forma de importar librerías a TS
    - Agregar tipos a las variables y funciones
  + Ejercicio 2 (Partir de cero)
    - Definir funciones (con todos los tipos necesarios) para hacer lo siguiente:
    - Cargar un listado de palabras (por esta vez, usar el arreglo como variable global)
    - Insertar/eliminar/buscar/actualizar una palabra del listado
  + Ejercicio 3 (Adicional)
    - Agregar que la lista de palabras esté ordenada permanentemente (ayudarse con filminas de ordenamiento)
* Manejo de Archivos de Texto
  + Instalar paquete → npm install @types/node
  + Crear archivo ‘abc.txt’
  + Escribir adentro ‘hola como andas todo bien’



* Ejercicios - Fuera de Clase
  + Ejercicio 1
    - Crear proyecto NPM
    - Implementar sistema para cargar un arreglo a partir de archivo de texto
    - Subir a GitHub y pasar el link por Slack
  + Ejercicio 2
    - Idem ejercicio anterior pero que cargue matrices a partir de archivo de texto
* ***Ejercicios Adicionales:*** La idea es ir incorporando lo visto en clases anteriores
  + Iniciar proyecto NPM en carpeta nueva
  + Descargar las librerías necesarias
  + Cargar un listado de nombres a partir de un archivo de texto
  + Cargar un listado de colegios a partir de otro archivo de texto
  + A partir de ambos listados, saber en qué colegio va a votar una determinada persona
  + Subir a GitHub y avisar por Slack

**Clase 3.2.1**

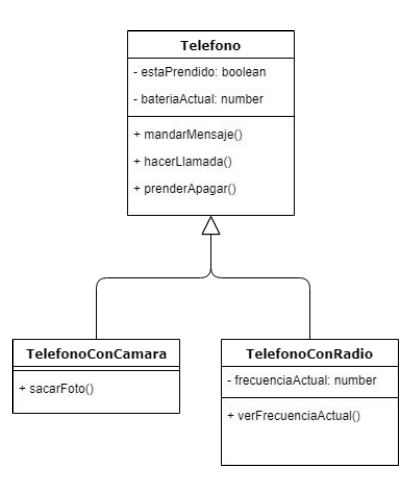
* Constructor - Demo en Clase
  + Implementación de la clase Telefono
* Ejercicios - En Clase En un mismo proyecto NPM
  + Ejercicio 1
    - Plantear la clase Auto de la forma en que se vió en la clase → especificando variables internas y métodos
    - Implementar en TypeScript
  + Ejercicio 2
    - Plantear la clase Monitor
    - Implementar en TypeScript
* Ejercicios - Fuera de Clase En un mismo proyecto NPM
  + Ejercicio 1
    - Usando la clase Auto: Implementar la clase RegistroAutomotor con métodos para consultar por un auto en un listado, borrar, actualizar y dar de alta
    - Partir de función ya implementada para leer archivos
  + Ejercicio 2
    - Implementar la clase Matriz. En vez de consultar los valores con los *corchetes*, se debe hacer (desde afuera) a través de un método → get(x, y)

**Clase 3.2.2**

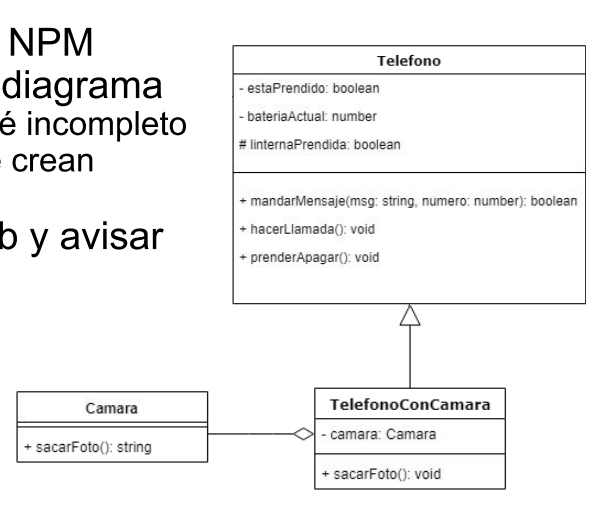
* Encapsulamiento - Demo en Clase
  + Implementación de clase utilizando modificadores de acceso → public/private
  + Forzar errores
  + Ver un poco de parámetros opcionales
* Ejercicios - En Clase
  + Para todos los ejercicios, crear proyecto NPM, subir a GitHub y avisar por Slack
  + Ejercicio 1
    - Agregar los conceptos vistos hoy al ejercicio 1 y 2 de la clase anterior
  + Ejercicio 2
    - Armar la clase Matriz (similar al de la clase anterior) pero aplicando conceptos de abstracción y encapsulamiento
  + Ejercicio 3
    - Implementar la clase Televisor y Decodificador
* Ejercicios - Fuera de Clase
  + Crear proyecto NPM y subir a Github
  + Implementar la clase LectorArchivos → partir del código facilitado en la clase anterior
  + Implementar la clase RegistroAutomotor: similar al ejercicio de la clase pasada, pero incorporando los conceptos nuevos, y la clase LectorArchivos
  + Mandar por Slack el link al repositorio de GitHub
    - *Aclaración*: no hay una sola forma de tener bien los ejercicios → lo que importa es saber justificar bien las decisiones que se tomen

**Clase 3.2.3**

* Ejercicios - En Clase - Usar los conceptos y recomendaciones vistas durante esta semana y la anterior
  + Armar una base de datos de libros
  + Hacer el planteo de las clases necesarias
  + Implementar la clase Libro
  + Implementar la clase GestorLibros → debe soportar
    - Inserter
    - Consultar
    - Modificar
    - eliminar libros (la entrada de información por teclado)
  + Luego incorporar en donde se crea necesario un mecanismo para leer libros desde un archivo de texto
  + Subir las cosas a GitHub y avisar por Slack
* Ejercicios - Fuera de Clase - Incorporar los conceptos y recomendaciones vistas durante esta semana y la anterior - Librerías NPM
  + Iniciar un proyecto NPM • Elegir una librería en https://www.npmjs.com/ e incorporarla en el proyecto
  + Modelar una clase con composiciones, que utilice dicha librería
  + Definir tarea NPM para compilar y correr los archivos necesarios
  + Subir proyecto a GitHub y avisar por Slack

**Clase 3.3.1**

* Demo en Clase
  + Partir de la clase Televisor en el repo de GitHub
* Ejercicios - En Clase
  + Implementar las clases y métodos que se muestran >
  + Agregar variables/métodos adicionales
  + Implementar cada clase en un archivo diferente • Ojo con la forma de hacer los import
  + Subir las cosas a GitHub y avisar por Slack
* Ejercicios - Fuera de Clase
  + Crear proyecto NPM • Subir proyecto a GitHub
  + Implementar Registro Automotor visto anteriormente, pero agregando soporte de motos y camiones usando herencia
  + Definir tarea NPM para compilar y correr los archivos necesarios
  + Enviar por Slack el link al repositorio de GitHub

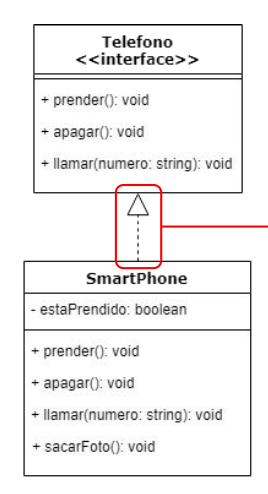


**Clase 3.3.2**

* Ejercicios - En Clase
  + Iniciar proyecto NPM
  + Implementar el diagrama
  + Puede que esté incompleto
  + Agregar lo que crean necesario
  + Subirlo a GitHub y avisar por Slack
* Ejercicios - Fuera de Clase
  + Ejercicio 1 (empleando draw.io)
    - Elegir tres ejercicios realizados anteriormente de los fuera de clase y plantear el diagrama de clase
  + Ejercicio 2 (empleando draw.io, NPM y GitHub)
    - Plantear un diagrama de clase con los siguientes requisitos e implementar
      * Herencia
      * Composición
      * Variables protected
    - Avisar por Slack cuando tengan las cosas 🤙

**Clase 3.3.3**

* Ejercicios - En Clase
  + Ejercicio 1
    - Iniciar proyecto NPM y Git
    - Plantear diagrama de clases para clases Auto y AutoCarreras redefiniendo métodos
    - Implementar el código
    - Subir a GitHub y avisar por Slack
  + Ejercicio 2
    - Idem ejercicio anterior, pero proponiendo un ejercicio que emplee una clase padre y dos clases hijas → usando polimorfismo
    - Subir a GitHub y avisar por Slack
* Ejercicios - Fuera de Clase
  + Iniciar proyecto NPM y Git
  + Plantear un diagrama de clase con los siguientes requisitos e implementar
    - Herencia
    - Composición
    - Variables protected
    - Polimorfismo
    - Métodos privados
  + Definir tarea NPM para compilar y correr los archivos necesarios
  + Subir proyecto a GitHub • Mandar por Slack el link al proyecto 🤙



**Clase 3.4.1**

* Ejercicios - En Clase (1)
  + Ejercicio 1
    - Iniciar proyecto NPM
    - Implementar el siguiente diagrama de clases >
    - Subir a GitHub y avisar por Slack
  + Ejercicio 2
    - Iniciar proyecto NPM
    - Implementar las siguientes clases con las variables y métodos que crea necesarios:
      * AutoCiudad
      * AutoDeportivo
      * Camioneta
    - Abstraer elementos en común entre dichas clases → pasarlos a una clase abstracta, y que las tres clases extiendan de ella
* Ejercicios - Fuera de Clase
  + A partir del código provisto, implementar una base de datos básica → CRUD (create, read, update, delete)
  + Parte 1
    - Generar diagrama de clases empleando interfaces o clases abstractas
    - Implementar sistema
  + Parte 2
    - El sistema tiene que ser capaz de persistir los datos → leer/escribir (investigar fs.writeFile) archivos TXT
    - La idea es que no se pierdan los datos cuando termine la ejecución del programa

**Clase 3.4.2**

* Ejercicios - En Clase
  + Para todos los ejercicios plantear diagrama de clases. También usar tareas NPM y GitHub
  + Ejercicio 1
    - Implementar un sistema de archivos
    - Investigar patrón *composite*
  + Ejercicio 2
    - Extender clase Auto implementada anteriormente, para que soporte seis tipos de datos más → todos pasados en el constructor
    - Investigar patrón *builder*
* Ejercicios - Fuera de Clase
  + Implementar un logger que incorpore al sistema de gestión de archivos
    - Investigar patrón *singleton*
  + Incluir un archivo de texto que explique las razones de por qué tomaron las decisiones que tomaron
  + NPM, GitHub y draw.io

**Clase 3.4.3**

* Ejercicios - En Clase
  + Tomar tres ejercicios a elección que hayan hecho antes, y modificar la validación de los datos para hacerla empleando errores
* Ejercicios - Fuera de Clase
  + Ejercicio 1
    - Agregar soporte de errores a la base de datos básica de la clase 1
  + Ejercicio 2
    - A partir del soporte de errores de la base de datos, agregar a dicho soporte manejo de *errores propios*

**Clase 3.5.1**

* Ejercicios
  + Partiendo de la base de datos armada en las últimas clases
  + Generar diagrama de clases usando draw.io
  + Escribir un comentario de cada uno de los métodos implementados → el objetivo es ir ejercitando la justificación de las decisiones de diseño tomadas
  + Con respecto a las clases, escribir un comentario para cada una, indicando la responsabilidad de cada clase, y la funcionalidad que provee

**Clase 3.5.2**

* Parte 1 - Diseño de Sistemas
  + Implementar un sistema de control de stock para un kiosco. Dicho sistema debe tener un listado de los elementos disponibles para vender, y un listado de los elementos vendidos
    - Tener en cuenta que cada ítem o elemento tiene un costo asociado
    - El sistema debe poder cargar de un archivo los items a vender
    - El sistema debe poder bajar a un archivo los ítems vendidos
* Parte 2 – Documentación
  + A partir del código TypeScript, plantear el diagrama de clases asociado
  + Tener en cuenta las relaciones entre clases

• Herencia • Implementación • Composición

* + Tener en cuenta los modificadores de acceso en las variables y métodos

• + → public • - → private • # → protected

* + Partir del código disponible en el repo

• semana-5/documentacion-1

* Parte 3 - Preguntas Teóricas
  + Responder las preguntas en un TXT y subirlo a GitHub
    - Enumerar tres funcionalidades de NPM y describirlas
    - ¿Cuál es el beneficio de usar un lenguaje con tipos?
    - ¿A qué se le llama variable interna? ¿Por qué internas?
    - Explicar la diferencia entre composición y herencia
    - Explicar el mecanismo que provee TypeScript para manejar casos en donde los parámetros que le llegan a un método son inválidos

**Clase 3.5.3**

* Parte 1 - Diseño de Sistemas
  + Implementar un sistema de gestión de legajos para un colegio. Dicho sistema debe llevar registro de los alumnos. A su vez, cada alumno, además de sus datos personales, tiene un listado de exámenes rendidos. El sistema debe proveer la siguiente funcionalidad:
    - Obtener el promedio general de un determinado alumno
    - Obtener el promedio de todos los alumnos inscriptos en el sistema
    - El alta de los alumnos a elección: a partir de un archivo, a partir de teclado, o creandolos directo en el código
* Parte 2 – Documentación
  + A partir del código TypeScript, plantear el diagrama de clases asociado
  + Tener en cuenta las relaciones entre clases

• Herencia • Implementación • Composición

* + Tener en cuenta los modificadores de acceso en las variables y métodos

• + → public • - → private • # → protected

* + Partir del código disponible en el repo

• semana-5/documentacion-2

* Parte 3 - Preguntas Teóricas
  + Responder las preguntas en un TXT y subirlo a GitHub
    - ¿Para qué sirve una task de NPM?
    - ¿Qué pasa cuando se corre el comando “tsc”?
    - ¿Cómo está organizada una clase?
    - ¿Es correcto que un método auxiliar de una clase sea público? ¿Por qué?
    - Enumerar las características de una interfaz
    - Nombrar al menos dos patrones de diseño, y en qué casos se utilizan
    - ¿Cómo se hace para definir un error nuevo en TypeScript?