Curso CFP

# CFP Programador full-stack

*Repaso + Diagramas de Clase*

# Agenda

- Repaso
  - Herencia
  - Herencia vs. Composición
- Nueva forma de plantear los diseños
- Diagramas de Clase
- Representación de Clases
- Representación de Métodos
- Modificadores de Acceso
- Relaciones entre Clases
  - Composición
  - Herencia
- Recomendaciones
- Ejercicios

# Repaso - Herencia

- Cuando tenemos dos clases con variables y métodos similares
  - Las cosas en común ponerlas en una superclase
  - Las dos clases originales extenderán la superclase
- Sirve para evitar que dupliquemos el código
  - Por lo tanto → para escribir menos
  - Código más claro y prolijo
- Variables *protected* → privadas salvo para subclases
- El constructor de la subclase *debe* incluir una llamada al constructor de la superclase → *super*
  - En caso de no incluirla → *tsc* se queja

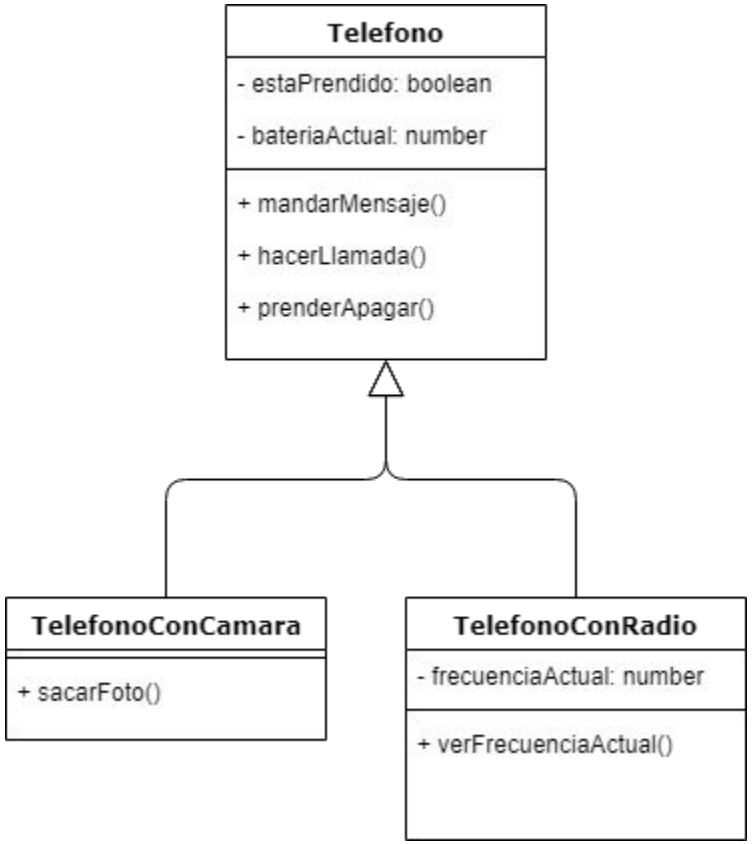
# Repaso - Herencia vs. Composición

- Si queremos usar toda la funcionalidad de una clase, y agregar más → herencia
  - Recordar que herencia hace que la subclase posea todas las cosas *public* y *protected* de la superclase
- Si queremos usar solamente algunas cosas de una clase → composición
  - Recordar que si finalmente terminamos llamando a todos (o casi) los métodos de dicha clase, probablemente sea mejor usar herencia
- Evitar *forzar* la herencia → un auto y un reloj pueden compartir un número de serie, pero no tienen nada que ver entre sí

# Nueva forma de plantear los diseños

- Hasta el momento, entendemos lo que un código hace a partir de *leer* el mismo
  - En caso de tener códigos muy largos, esto se puede complicar
- En varias ocasiones vimos diagramas para entender cómo organizar/diseñar las clases
- A partir de la lectura del *diagrama de clases*, podemos darnos una idea de cómo está organizado el código
- Normalmente planteamos el diagrama *antes* de pasar al código → asegurarse en todo momento que el código respete al diagrama

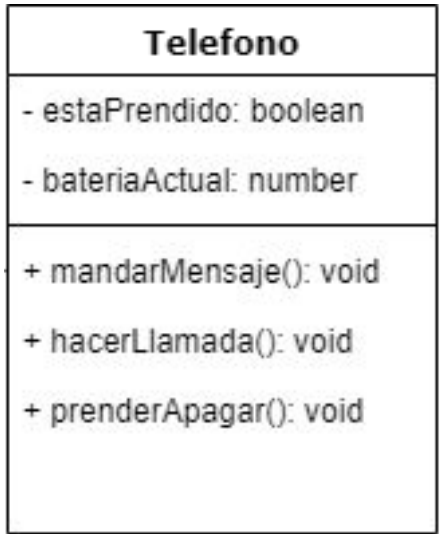
# Diagramas de Clase




- Se usan para representar código
- La idea es lograr que el diagrama explique lo que hace un sistema
  - Evitando tener que ir al código
- Recordar que programar se hace más fácil si antes tenemos un diseño de la solución
  - Diagramas

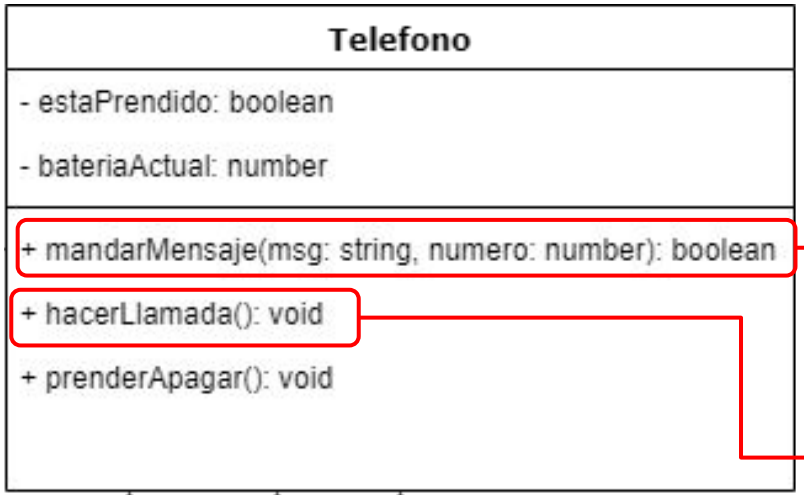
# Representación de Clases

- Las clases se representan con un recuadro compuesto por tres secciones
- La primera indica el *nombre* de la clase
- La segunda indica las *variables internas*
- La tercera indica los *métodos*
- Los modificadores de acceso se representan igual tanto para variables/métodos
  - private → -
  - public → +
  - protected → #
- Los tipos también se especifican
  - En variables es el propio tipo
  - En métodos es el tipo de valor que devuelve



# Representación de Métodos

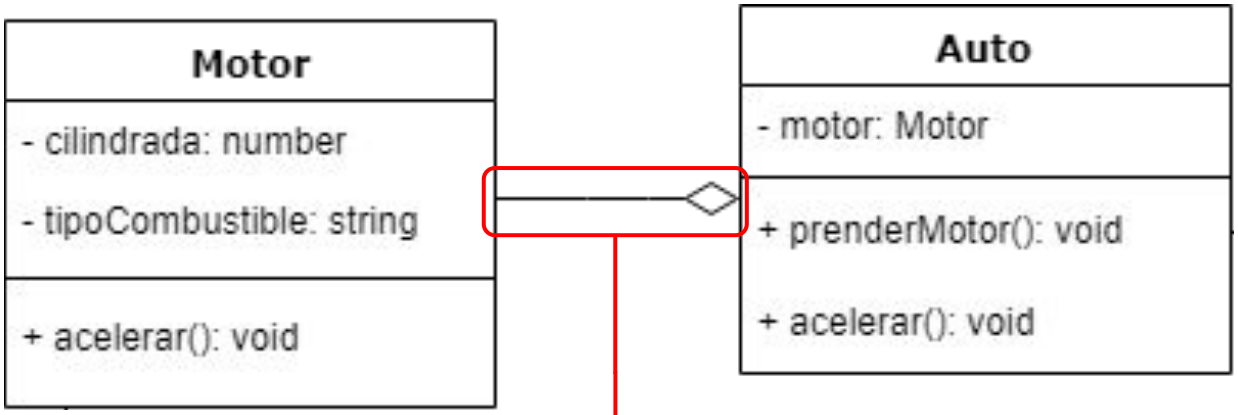
- En los métodos también se ponen los parámetros
- Se especifican las variables y el tipo




- **Método público**
- **Retorna un boolean**
- **Dos parámetros**
  - **Texto**
  - **Numérico**

- **Método público**
- **No retorna nada**
- **No tiene parámetros**

# Relaciones entre Clases (1)

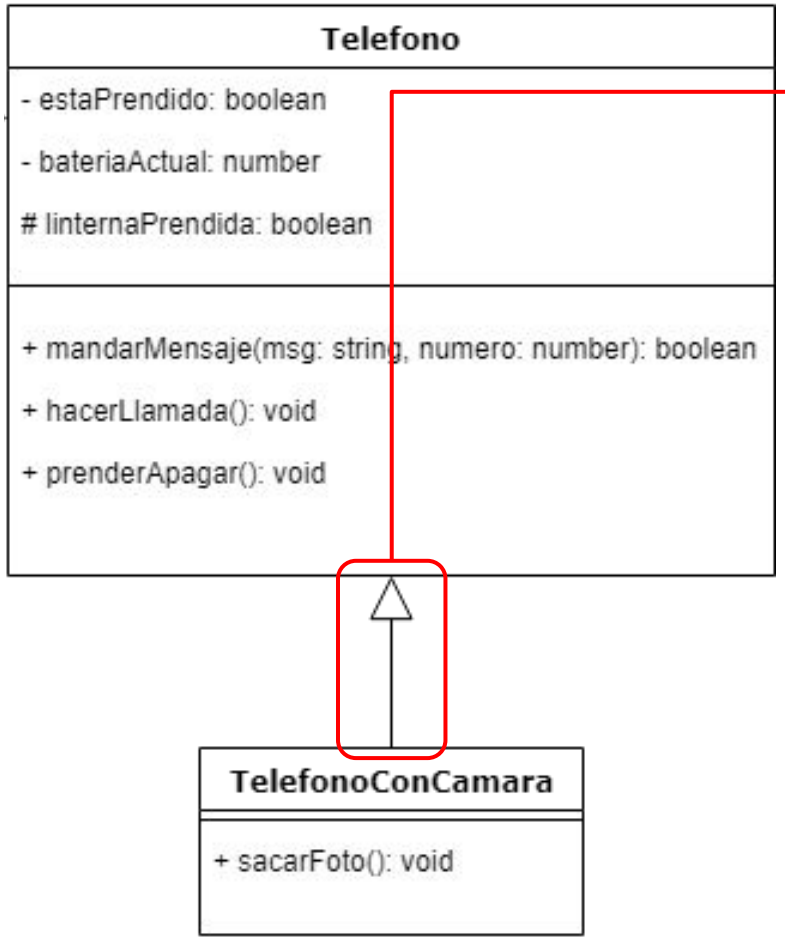



**COMPOSICIÓN**

**La flecha se lee desde el *origen* hacia el *final***

**La clase Motor *compone* a la clase Auto**
**La clase Auto *está compuesta* por la clase Motor**

# Relaciones entre Clases (2)




**HERENCIA**

**La flecha se lee desde el *origen* hacia el *final***

**La clase TelefonoConCamara *hereda de* la clase Telefono**

# Recomendaciones

- El diagrama *siempre* debe ser acorde al código
- Plantear el diagrama antes de escribir el código
  - Normalmente aparecen problemas durante la implementación, que no fueron tenidos en cuenta en el diagrama
  - No hay ningún problema en ir acomodando el diagrama durante la implementación → lo ideal es tratar de reducir al mínimo estas situaciones
- Cuanto más tiempo le dediquen al diagrama, más fácil les va a resultar escribir el código

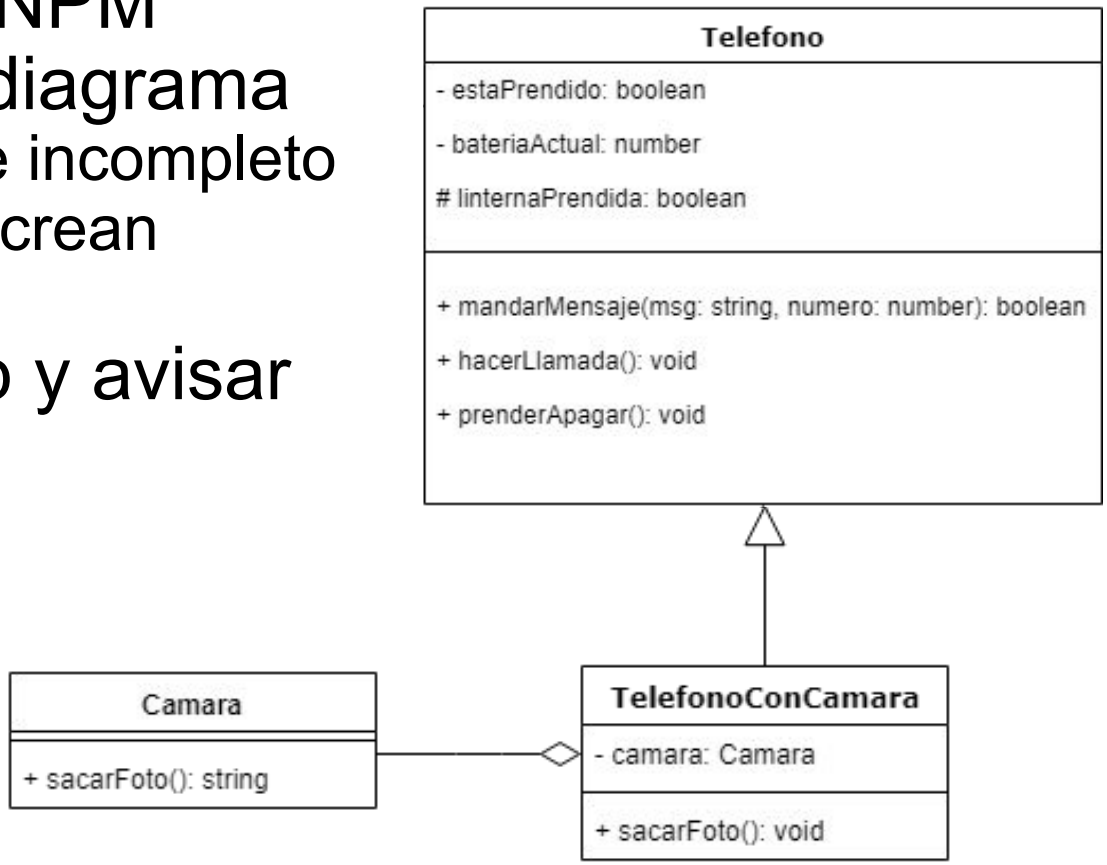
Curso CFP

# CFP Programador full-stack

*Ejercicios*

# 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 👇