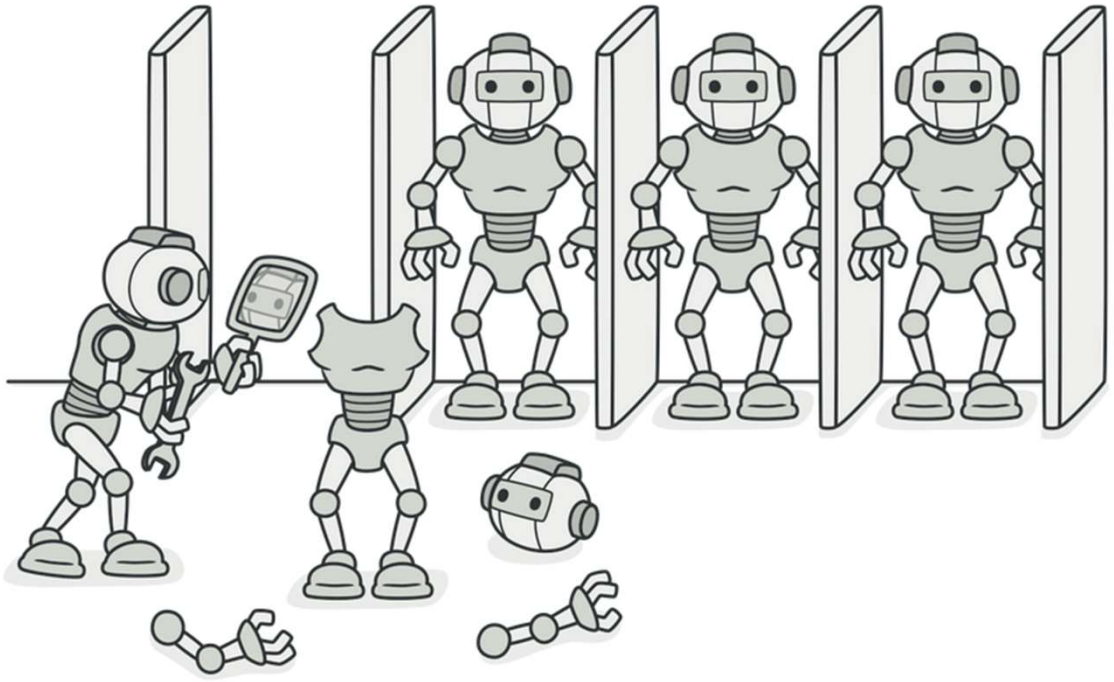
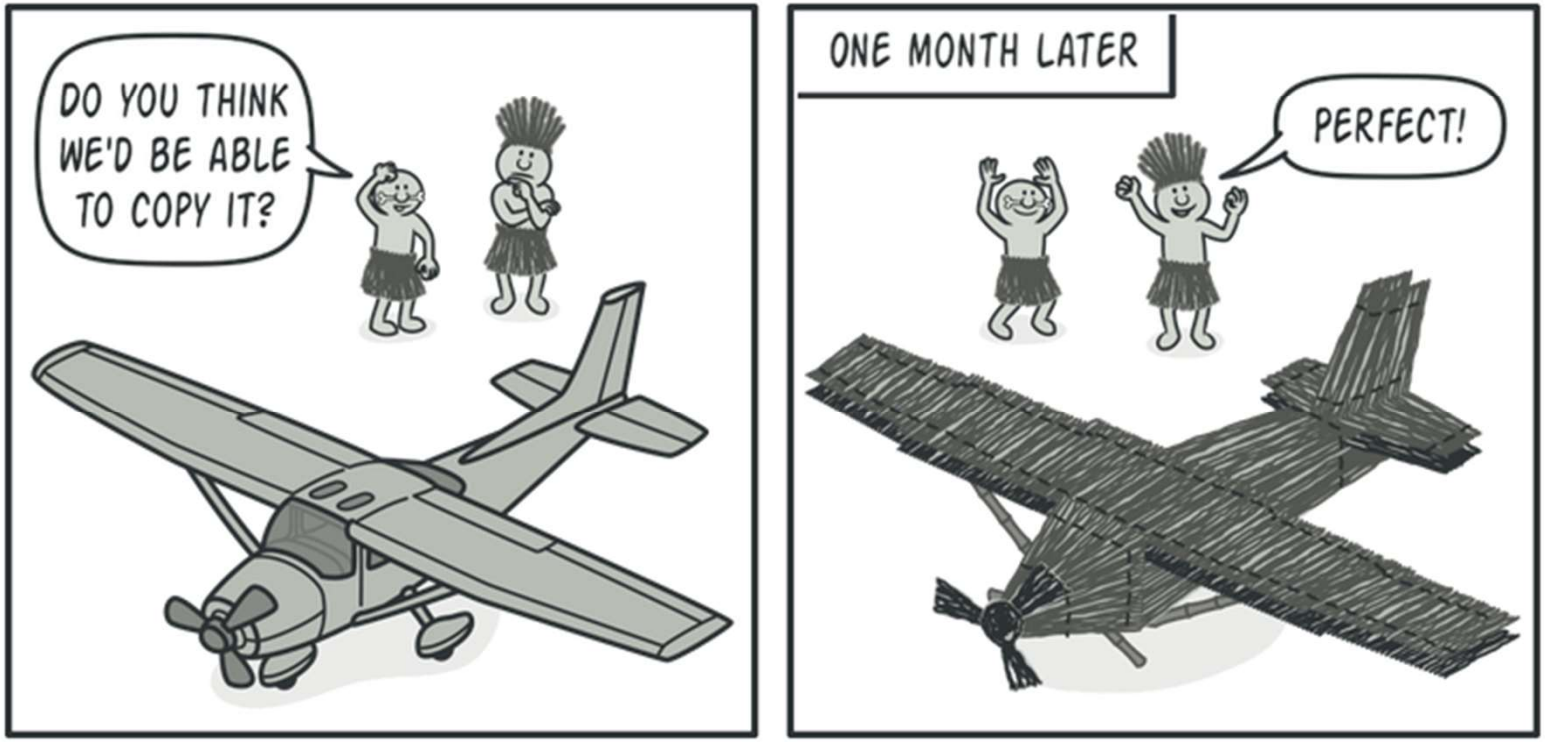
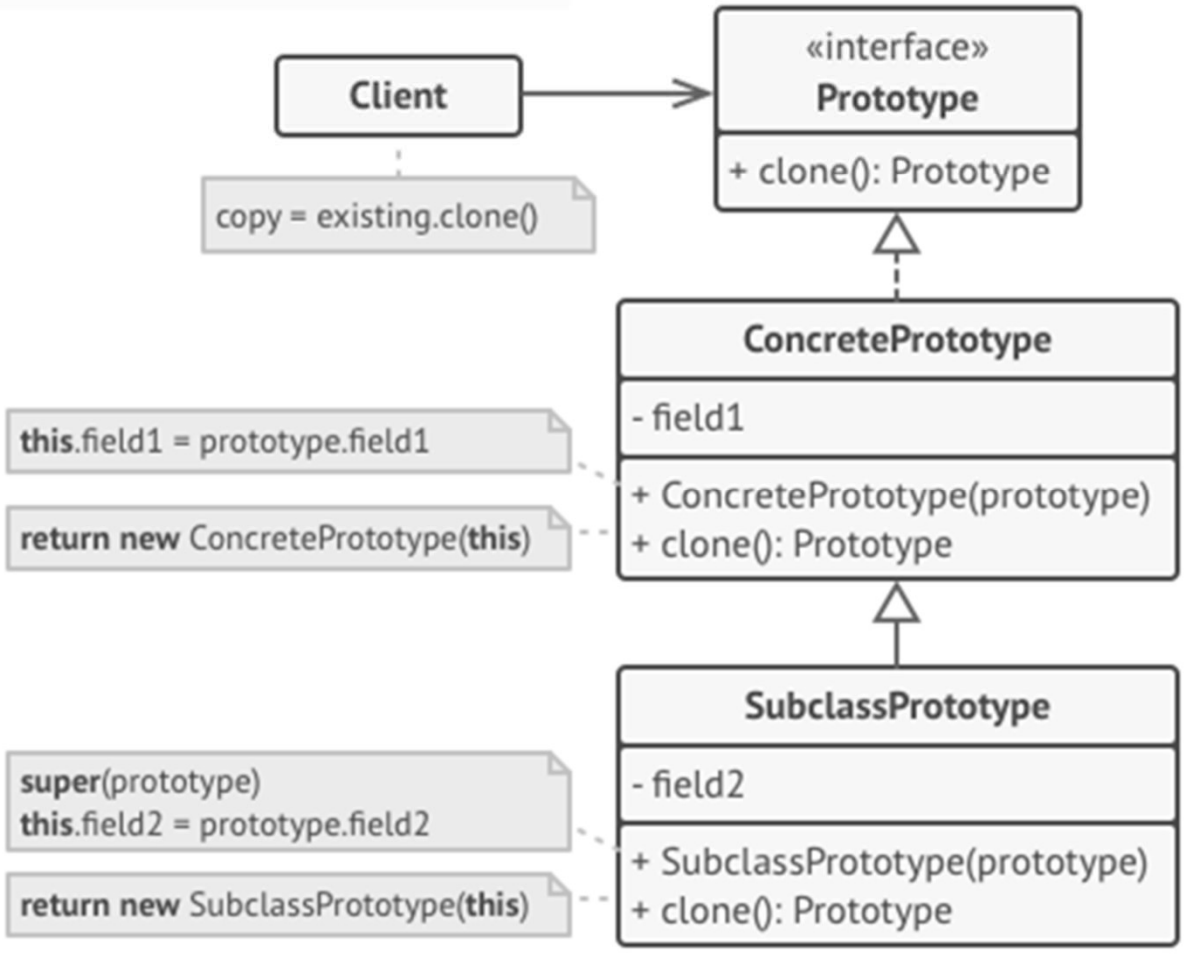
# Прототип






*Скопіювати об'єкт «ззовні»* ***не*** *завжди можливо.*


*Попередньо створені прототипи можуть бути альтернативою підкласу.*




2 Клас **Concrete Prototype** реалізує метод клонування. Окрім копіювання вихідних даних об'єкта до клону, цей метод також може обробляти деякі граничні випадки процесу клонування, пов'язані з клонуванням пов'язаних об'єктів, розплутуванням рекурсивних залежностей тощо.

# Плюси і мінуси

- Ви можете клонувати об’єкти без зв’язку з їхніми конкретними класами.
- Ви можете позбутися повторного коду ініціалізації на користь клонування попередньо зібраних прототипів.
- Ви можете виготовляти складні об’єкти зручніше.
- Ви отримуєте альтернативу успадкуванню, коли маєте справу з налаштуваннями конфігурації для складних об’єктів.

- Клонування складних об’єктів, які мають циклічні посилання, може бути дуже складним.

# Поєднання з іншими патернами

- Багато проектів починаються з використання **Factory Method** (менш складного та більш настроюваного за допомогою підкласів) і розвиваються до **Abstract Factory , Prototype** або **Builder** (більш гнучкого, але складнішого).

- Класи **Abstract Factor**у часто базуються на наборі **Factory Methods** , але ви також можете використовувати **Prototype** для створення методів цих класів.

- Прототип може допомогти, коли вам потрібно зберегти копії команд в історію.

- Дизайни, які активно використовують **Composite та Decorator**, часто можуть виграти від використання **Prototype** . Застосування шаблону дозволяє клонувати складні структури замість того, щоб реконструювати їх з нуля.

- Прототип не заснований на спадкуванні, тому він не має своїх недоліків. З іншого боку, **Prototype** вимагає складної ініціалізації клонованого об'єкта. Фабричний метод заснований на успадкуванні, але не потребує кроку ініціалізації.