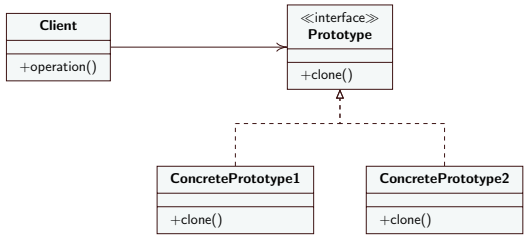
Prototype

- **Intento**: specificare i tipi di oggetti da creare usando una istanza di un prototipo, e creare nuovi oggetti copiando questo prototipo
- Motivazione
  - Durante la creazione di un nuovo oggetto può essere necessario avere uno stato iniziale che è simile a quello di un altro oggetto esistente, e che quest'ultimo ha ottenuto per mezzo di operazioni eseguite su esso
  - Quando si crea un oggetto bisogna indicare una specifica classe, e così la classe che chiede l'istanza è strettamente accoppiata con la classe da istanziare
  - Lo stretto accoppiamento fra classi da istanziare e chi istanzia rende difficile l'aggiunta di nuovi tipi, per es. nel Factory Method bisogna creare una gerarchia di Creator per aggiungere nuovi tipi

Prototype: Soluzione

- Struttura



- Partecipanti
  - **Prototype** definisce l'interfaccia per le operazioni comuni e dichiara un metodo per clonare se stessa
  - **ConcretePrototype** è una classe che implementa l'operazione di clonazione
  - **Client** è una classe che crea nuovi oggetti tramite clonazione dell'istanza prototipo, quindi senza indicare la classe esplicitamente. Client conosce solo l'interfaccia Prototype

Prototype: Conseguenze

- Il client non conosce la classe usata, poiché il client usa l'interfaccia comune, quindi il numero di dipendenze del client è ridotto
- Il client può usare classi specifiche (sottoclassi) pur non conoscendone il nome
- Si può registrare un nuovo prototipo (o rimuovere un prototipo) a runtime
- Si possono specificare nuovi oggetti che tengono valori diversi, senza essere obbligati a creare nuove classi
- Un nuovo prototipo può essere usato per tenere una struttura diversa di oggetti (composizione). Gli oggetti della composizione sono attributi del prototipo, come in un Composite
- Per avendo tante classi diverse da poter usare (i ConcretePrototype), non si ha necessità di implementare una gerarchia di Creator (come invece avviene per il Factory Method), in quanto un nuovo oggetto si ottiene tramite clonazione
- E' possibile creare una configurazione (un prototipo) dinamicamente

Prototype: Implementazione

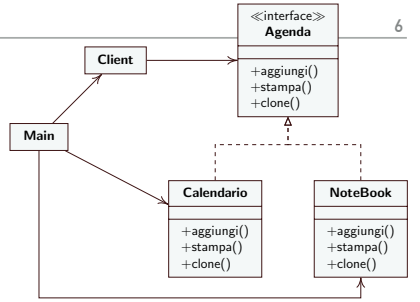
- L'uso di un **manager di prototipi** (che è un registro di istanze) permette ai client di immagazzinare e recuperare i prototipi, prima di clonarli
- L'implementazione dell'operazione di clonazione può essere la parte più difficile quando le strutture contengono riferimenti circolari
  - Sebbene un'operazione di clonazione potrebbe essere fornita dal linguaggio di programmazione, l'operazione di clonazione potrebbe effettuare una **copia superficiale (shallow copy)** anziché una **copia profonda (deep copy)**
  - In una shallow copy i riferimenti tenuti negli attributi dell'oggetto da clonare sarebbero condivisi fra originale e clone, e questo spesso non è sufficiente, quindi occorre una deep copy
- I cloni potrebbero necessitare di essere inizializzati: la classe prototipo potrebbe definire operazioni per impostare dati

Prototype: Esempio

- Si vuol tenere un'agenda di impegni, varie voci possono essere aggiunte all'agenda
- Lo stato dell'agenda è l'insieme di impegni presi
- Una nuova istanza dell'agenda deve poter avere accesso agli impegni precedentemente inseriti
- In modo simile, si vogliono inserire note in un quaderno e accedere alle note precedenti quando si ha una nuova istanza di quaderno

Prototype: Esempio

- Agenda è un Prototype: definisce l'operazione clone e le operazioni per le classi concrete
- Calendario e NoteBook sono ConcretePrototype, implementano le operazioni (compresa la clone)
- La classe Client chiama l'operazione di clonazione




```
public interface Agenda {
    public void aggiungi(String evento, LocalDateTime data);
    public void stampa();
    public Agenda clone();
}
```

```
public class NoteBook implements Agenda {
    private List<String> note = new ArrayList<>();
    @Override public void aggiungi(String evento,
                                   LocalDateTime t) {
        note.add(evento + ", " + t.getDayOfWeek() + " "
                 + t.getHour() + ":" + t.getMinute());
    }
    @Override public void stampa() {
        note.forEach(e -> System.out.println(e));
    }
    @Override public Agenda clone() {
        // deep copy
        List<String> n = new ArrayList<>();
        n.addAll(note);
        NoteBook notenew = new NoteBook();
        notenew.note = n;
        return notenew;
    }
}
```