Prototype

A creational pattern

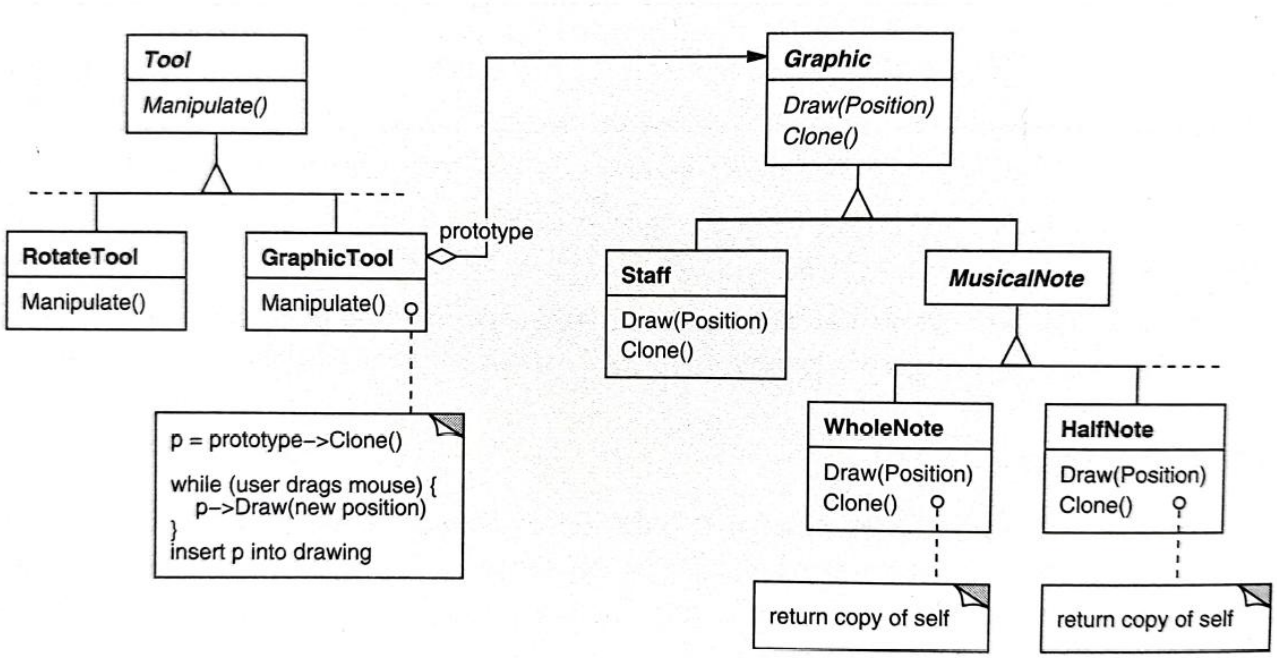
Learning goals

1. Learn the idea, structure, and Java implementation of the Prototype design pattern.
2. Learn to apply the Prototype DP in your own programming.

Idea of Prototype

- Allows creating new objects by copying existing instances.
  - A copied instance is called a prototype.
  - The prototype can then be further tailored to suit the needs of the creator.
- Situations for Prototype use:
  - When creating an object is more expensive than copying an existing one.
  - When the system should be independent of how its objects are created, composed, and represented.

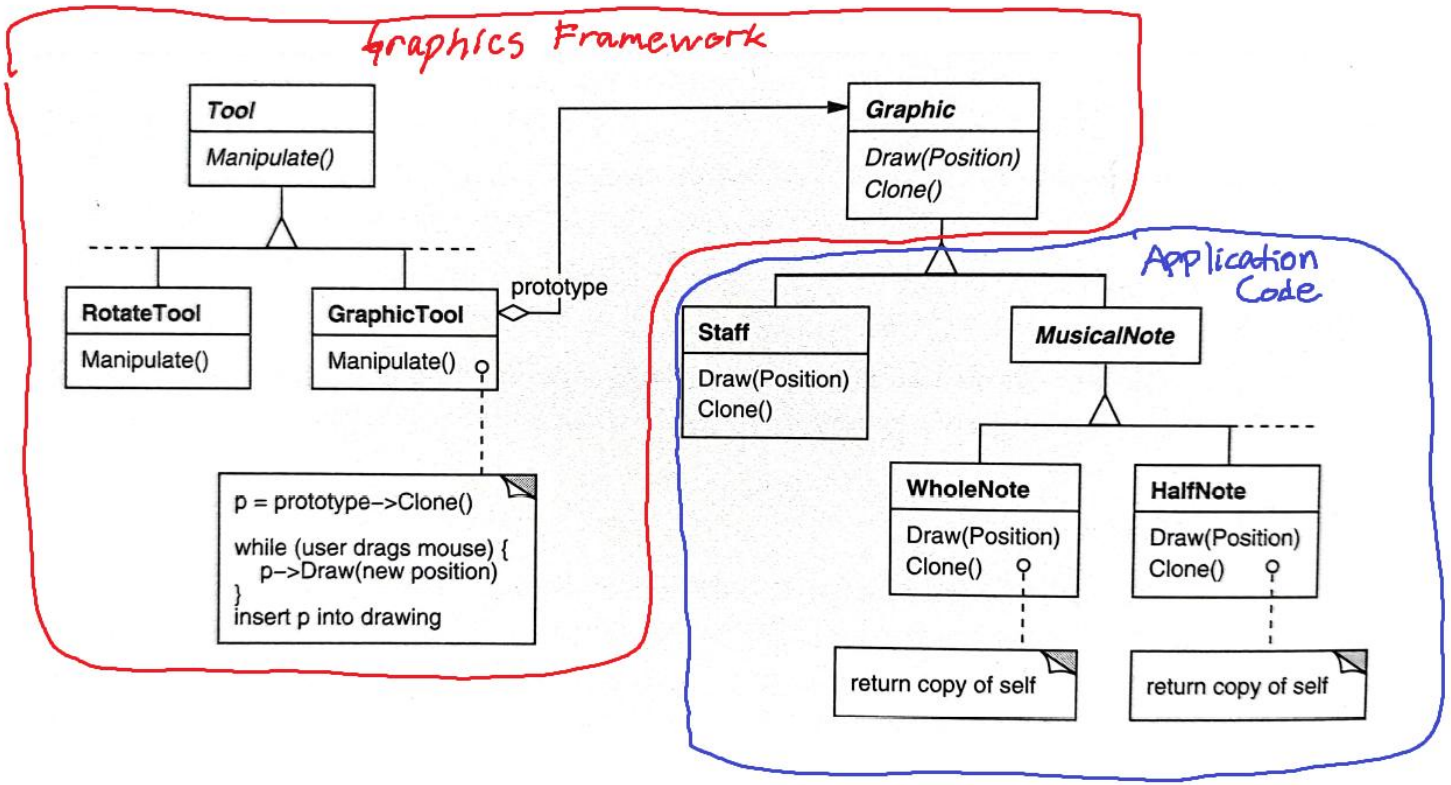
Example: music objects



- The graphics editor framework is used to draw notes and staves.

Image: Gamma et al., Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley Longman (1995), p. 118

Example: music objects



- The Framework part doesn't know of the Application code there may exist.

Image: Gamma et al., Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley Longman (1995), p. 118

Example: music objects

- The Graphic tool is able to handle Graphic objects.
- In the example, the manipulation method in the GraphicTool class needs to make a copy of the object right before dragging.
  - Challenge: The class doesn't know of the class, the instance of which it should create.
- Solution: The Graphic interfaces declares a clone() method for cloning an object, and the GraphicTool object calls it.
- The clone() method clones an existing object, whatever its type, in the way specified in the concrete implementation (e.g. HalfNote).

General structure

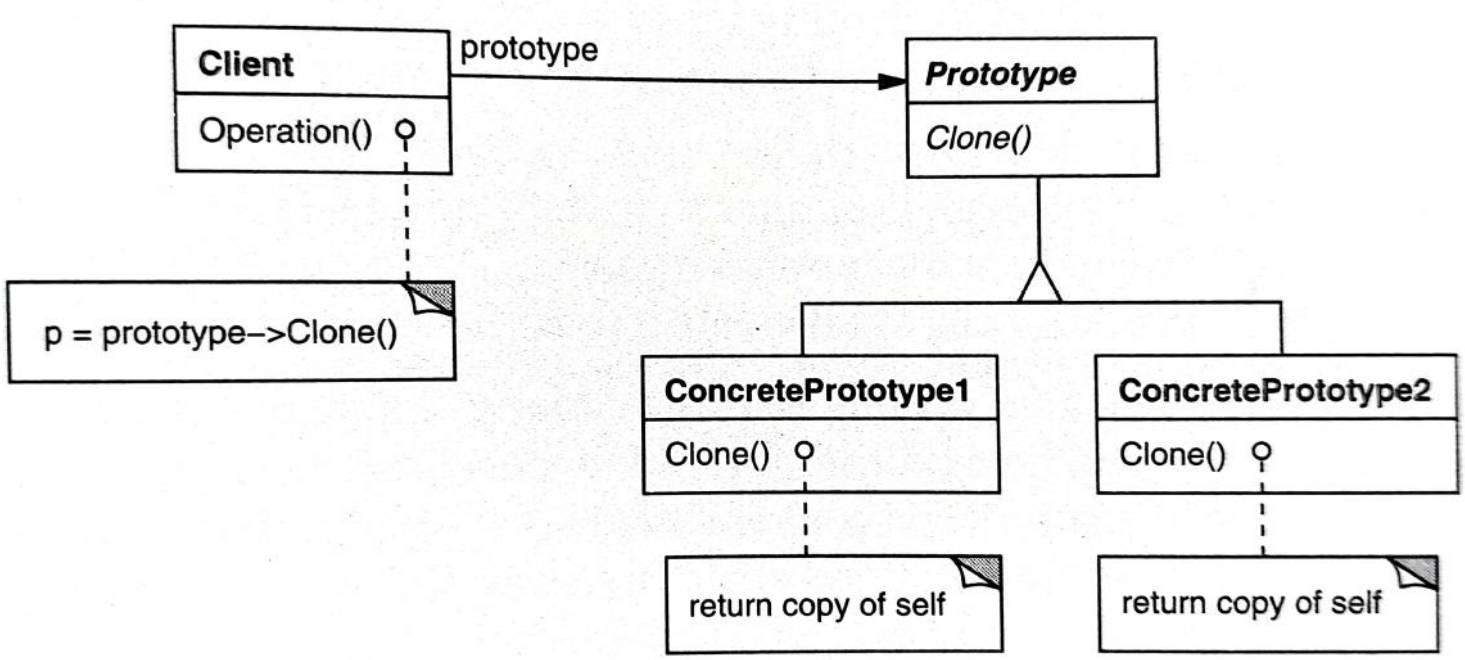


Image: Gamma et al., Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley Longman (1995), p. 119

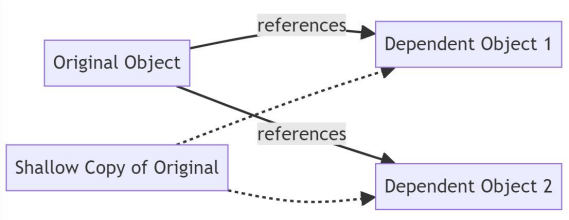
Roles

- **Prototype**: Declares an interface for cloning itself.
- **Concrete Prototype**: Implements the interface to clone itself.
- **Client**: Creates a new object by asking a prototype to clone itself.
  - This way, it avoids building a new object from scratch.

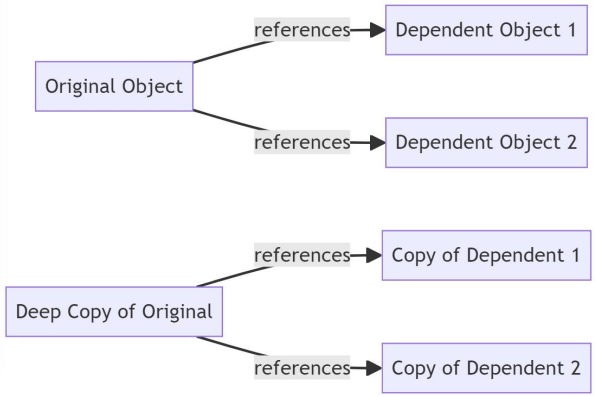
Deep and shallow copying

- Just as with the Memento pattern, the cloned object may have references to different objects.
- In shallow copying, only references to those object are copied.
- In deep copying, the states of the dependent objects are copied into new objects.

Shallow-copied clone



Deep-copied clone



Practical issues

- Complexity in Object Cloning:
  - Deep versus shallow copying can be a source of confusion.
  - Managing the cloning of objects with circular references or complex graphs.
- In Java, the application of the Prototype interface may differ:
  - There is a Cloneable interface (java.lang.Cloneable).
    - It doesn't declare a clone() method – or any methods at all.
  - The clone() method is implemented in the Object class.
    - When a class implements Cloneable, it permits Object.clone() to do its job.
  - Thus, it may not be necessary to create a custom Prototype interface.
- Cloning large objects can be costly in terms of performance. Unnecessary cloning can lead to memory issues.
- If the shallow-cloned object contains mutable fields, it can lead to unexpected side effects.
- Prototype requires that the classes are designed around the cloning functionality.
- May make testing more complicated.