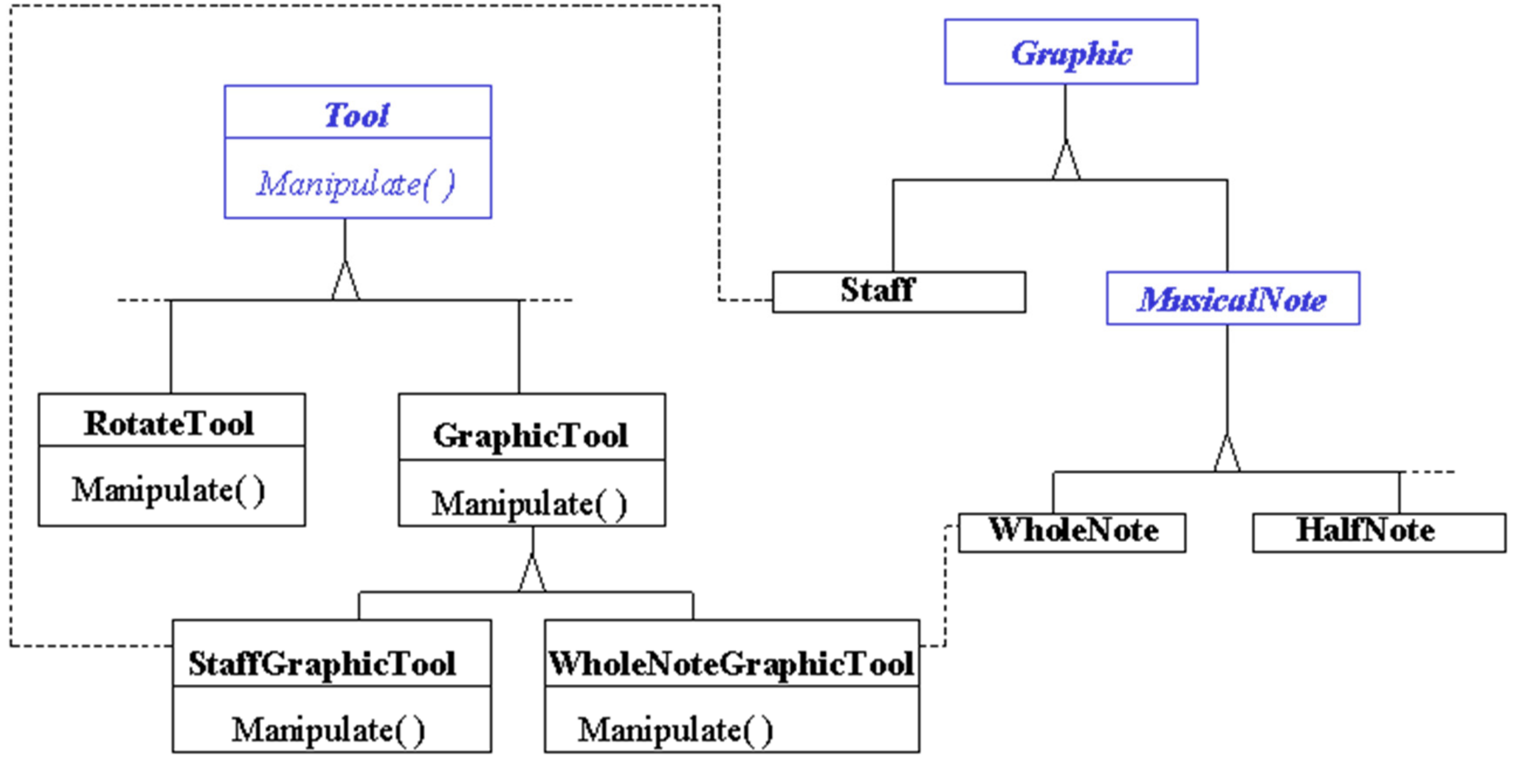
Tecniche di Progettazione: Design Patterns

GoF: Prototype

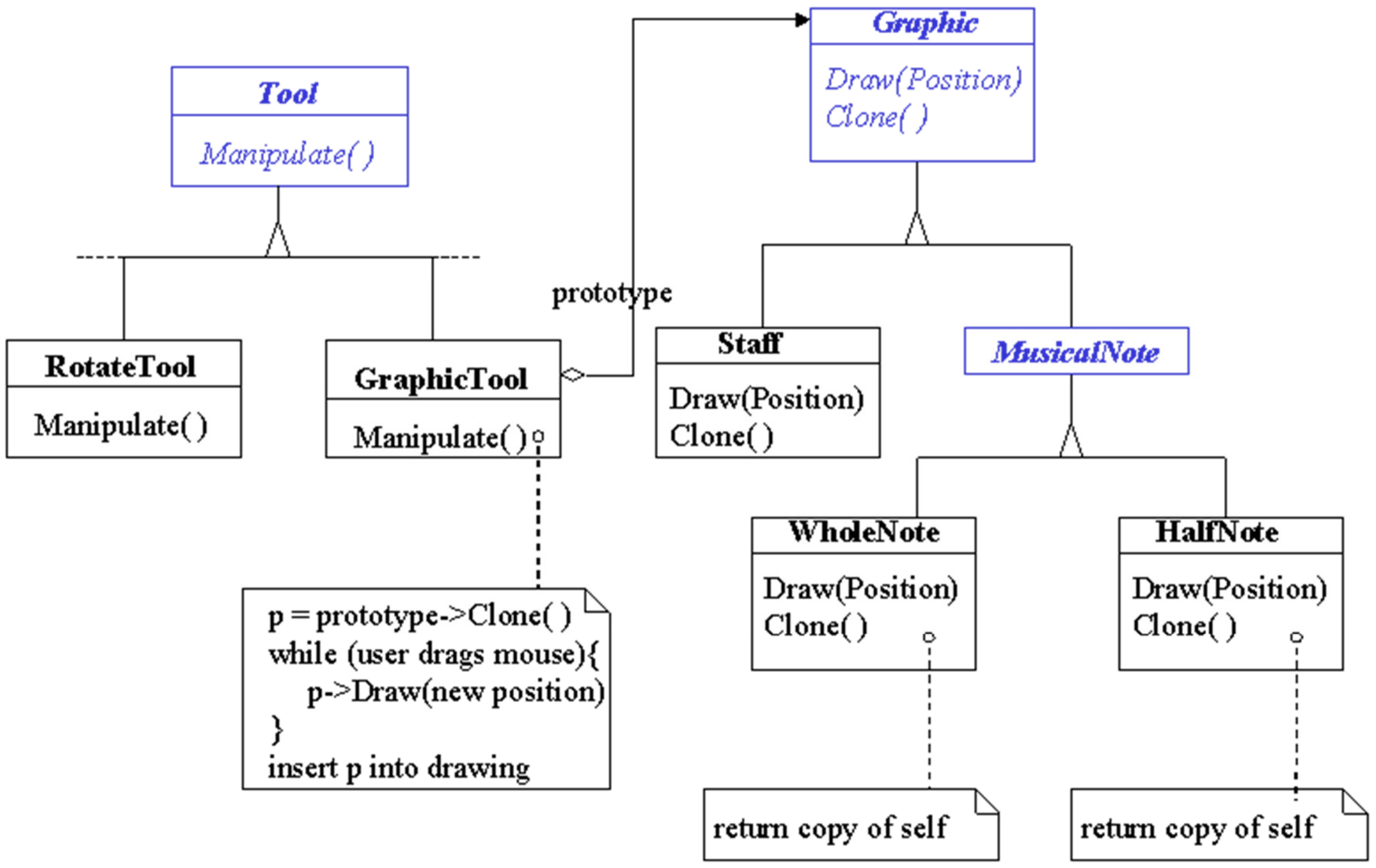
Prototype Pattern

- A creational pattern
- Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype

Problem


Tool
Manipulate( )
Graphic
Staff
MusicalNote
RotateTool
Manipulate()
GraphicTool
Manipulate()
WholeNote
HalfNote
StaffGraphicTool
Manipulate()
WholeNoteGraphicTool
Manipulate()

Prototype solution



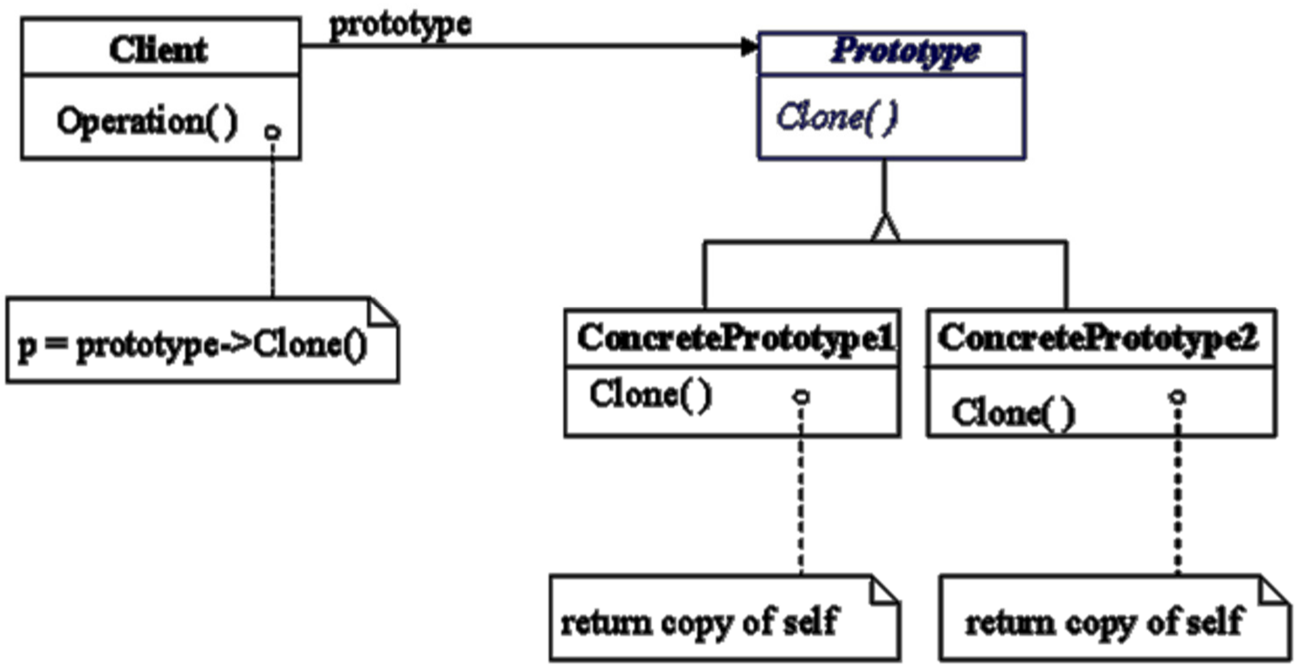
```
p = prototype->Clone( )
while (user drags mouse){
    p->Draw(new position)
}
insert p into drawing
```

Structure & Participants

Prototype(Graphic)
-declares an interface for cloning itself.

ConcretePrototype (Staff, WholeNote, HalfNote)
-implements an operation for cloning itself.

Client(GraphicalTool)
- creates a new object by asking a prototype to clone itself.



java.lang Class Object
protected Object **clone**() throws CloneNotSupportedException

Creates and returns a copy of this object. The precise meaning of "copy" may depend on the class of the object. The general intent is that, for any object x, the expression:

x.clone() != x

will be true, and that the expression:

x.clone().getClass() == x.getClass()

will be true, but these are not absolute requirements. While it is typically the case that:

x.clone().equals(x)

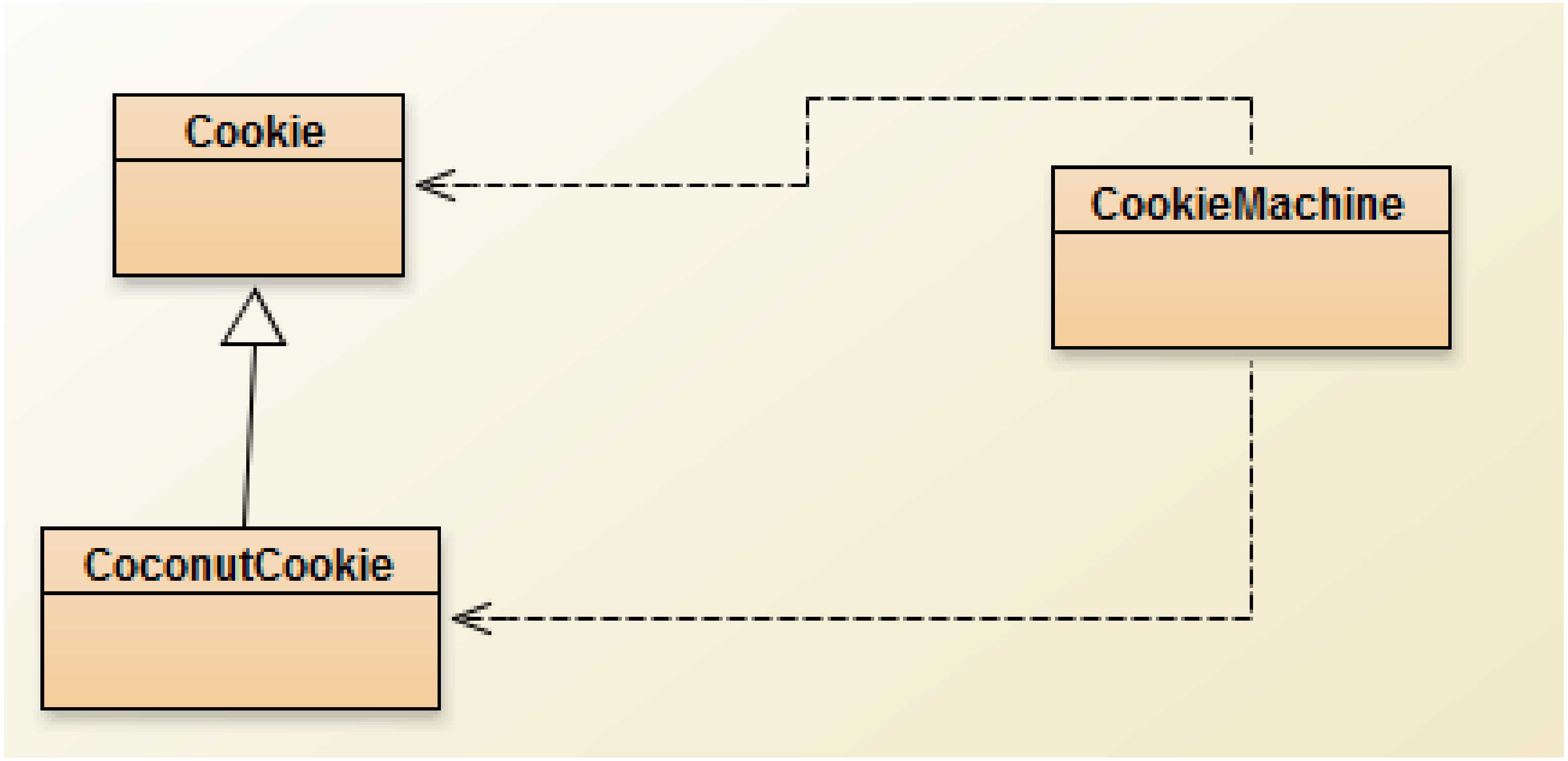
will be true, this is not an absolute requirement.

By convention, the returned object should be obtained by calling super.clone. If a class and all of its superclasses (except Object) obey this convention, it will be the case that x.clone().getClass() == x.getClass().

java.lang Class Object
protected Object **clone**() throws CloneNotSupportedException

- By convention, the object returned by this method should be independent of this object (which is being cloned).
- To achieve this independence, it may be necessary to modify one or more fields of the object returned by super.clone before returning it.
  - Typically, this means copying any mutable objects that comprise the internal "deep structure" of the object being cloned and replacing the references to these objects with references to the copies.
  - If a class contains only primitive fields or references to immutable objects, then it is usually the case that no fields in the object returned by super.clone need to be modified.

Ex. Cookies



Prototype Pattern Example code

```
public class Cookie implements Cloneable {
    public Object clone(){
        try{      Cookie copy = (Cookie)super.clone();
                  return copy;  }
        catch(CloneNotSupportedException e){
           e.printStackTrace();
           return null;       }
    }
     public void display(){
       System.out.println("I'm a cookie");    }
 }
```

Prototype Pattern Example code

```
public class CoconutCookie extends Cookie{

   public void display(){
      System.out.println("I'm a coconout cookie");
    }
    }
```

Prototype Pattern Example code

```
public class CookieMachine {
   private Cookie cookie;
   public CookieMachine(Cookie cookie) {
         this.cookie = cookie;  }
   public Cookie makeCookie() {
       return (Cookie)cookie.clone();   }
              // public Object clone() { }
   public static void main(String args[]){
         Cookie tempCookie =  null;
         Cookie prot = new
CoconutCookie();
         CookieMachine cm = new
CookieMachine(prot);
         for(int i=0; i<10; i++)
          {
             tempCookie = cm.makeCookie();
             tempCookie.display();
            }
         prot = new Cookie();
         cm = new CookieMachine(prot);
          for(int i=0; i<10; i++)
          {
             tempCookie = cm.makeCookie();
             tempCookie.display();
            }
     }
}
```

Output

```
I'm a coconout cookie
I'm a coconout cookie
I'm a coconout cookie
I'm a coconout cookie
I'm a coconout cookie
I'm a coconout cookie
I'm a coconout cookie
I'm a coconout cookie
I'm a coconout cookie
I'm a coconout cookie
I'm a cookie
I'm a cookie
I'm a cookie
I'm a cookie
I'm a cookie
I'm a cookie
I'm a cookie
I'm a cookie
I'm a cookie
I'm a cookie
```

Prototype Pattern

- When to Use
  - When product creation should be decoupled from system behavior
  - When to avoid subclasses of an object creator in the client application
  - When creating an instance of a class is time-consuming or complex in some way.

Consequences of Prototype Pattern

- Hides the concrete product classes from the client
- Adding/removing of prototypes at run-time
- Allows specifying new objects by varying values or structure
- Reducing the need for sub-classing

Drawbacks of Prototype Pattern

- It is built on the method .clone(), which could be complicated sometimes in terms of shallow copy and deep copy. Moreover, classes that have circular references to other classes cannot really be cloned.

No homework on this pattern