Prototype Pattern

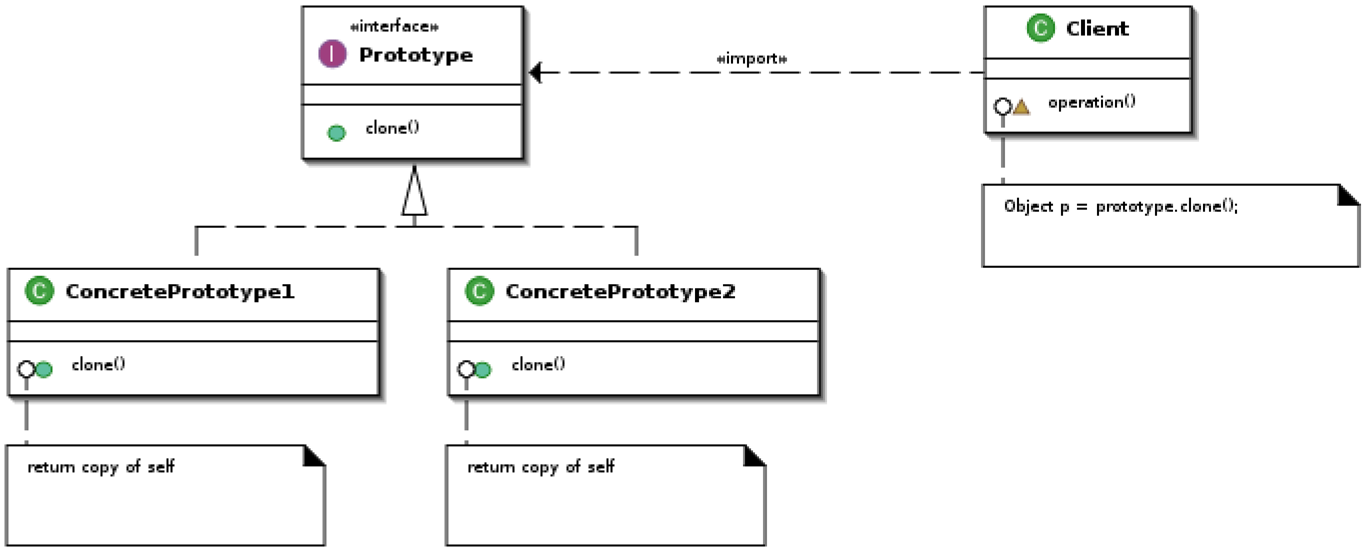
- Creational pattern that should be used when the type of objects to create is determined by a 'prototypical instance', which is cloned to produce new objects
- Avoids sub-classes of an object creator in the clien application (like abstract factory does)
- Biggest reason for using it (in Tom's opinion) – it avoids inherent cost of creating a new object in the standard way (via new) in cases where producing initial values for the fields of the object is costly (must be gathered from DB, calculated using time-consuming algorithm, etc.)

It's all about Clones!

Prototype Pattern

- In Java, utilize clone() to aid in this pattern (C# has MemberwiseClone)
- If you want to clone an object of a class it must implement the Cloneable interface
- Object's clone performs a shallow copy of existing object – if object has references to other mutable objects, you should add code to clone method of class you are cloning to perform a deep copy (by calling clone on the necessary objects that are fields of the current object)
- Deep copy can be done with serialization or even reflection and recursion
- BTW: clone is Java's copy constructor for those of you C++ minions

Prototype UML



Participants and Process

- <u>Client</u> - creates a new object by asking a prototype to clone itself.
- <u>Prototype</u> - declares an interface for cloning itself.
- <u>ConcretePrototype</u> - implements the operation for cloning itself.
- The process of cloning starts with an initialized and instantiated class.
  - The Client asks for a new object of that type and sends the request to the Prototype class.
  - A ConcretePrototype, depending of the type of object is needed, will handle the cloning through the Clone() method, making a new instance of itself.

Prototype Thoughts

- Prototype does not mean a 'not ready for prime time' version of the object, it means the standard version of the object after construction and initialization of the fields of that object
- Similar to Abstract Factory in that both use delegation (NOTE: Abstract Factory sometimes incorporates Prototype)
- Simplest to most complex in terms of the process of creating an object: Factory, Prototype, Builder, Abstract Factory

Code Example

```
public interface Prototype {
    public abstract Object clone ( );
}



public class ConcretePrototype implements Prototype {
    public Object clone() {
        return super.clone();
    }
}

public class Client {

    public static void main( String arg[] )
    {
        ConcretePrototype obj1= new ConcretePrototype ();
        ConcretePrototype obj2 = ConcretePrototype)obj1.clone();
    }

}
```

Applicability

- Use when
  - Prototype Pattern when a system should be independent of how its products are created, composed, and represented
  - Classes to be instantiated are specified at run-time
  - Avoiding the creation of a factory hierarchy is needed/desired (hierarchy can lead to explosion of classes)
  - It is more convenient to copy an existing instance than to create a new one.