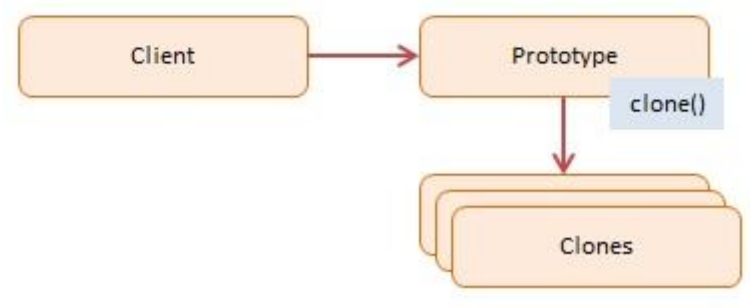
Summary

The Prototype Pattern creates new objects, but rather than creating non-initialized objects it returns objects that are initialized with values it copied from a prototype - or sample - object. The Prototype pattern is also referred to as the Properties pattern.

An example of where the Prototype pattern is useful is the initialization of business objects with values that match the default values in the database. The prototype object holds the default values that are copied over into a newly created business object.

Classical languages rarely use the Prototype pattern, but JavaScript being a prototypal language uses this pattern in the construction of new objects and their prototypes.

Diagram



Participants

The objects participating in this pattern are:

- **Client** -- In sample code: the run() function.
    - creates a new object by asking a prototype to clone itself
- **Prototype** -- In sample code: **CustomerPrototype**
    - creates an interfaces to clone itself
- **Clones** -- In sample code: **Customer**
    - the cloned objects that are being created

Sample code in JavaScript

In the sample code we have a CustomerPrototype object that clones objects given a prototype object. Its constructor function accepts a prototype of type Customer. Calling the clone method will generate a new Customer object with its property values initialized with the prototype values.

This is the classical implementation of the Prototype pattern, but JavaScript can do this far more effectively using its built-in prototype facility. You can learn about these and other techniques in ourJavaScript + jQuery Design Pattern Framework.

```
function CustomerPrototype(proto) {
    this.proto = proto;

    this.clone = function () {
        var customer = new Customer();

        customer.first = proto.first;
        customer.last = proto.last;
        customer.status = proto.status;

        return customer;
    };
}

function Customer(first, last, status) {

    this.first = first;
    this.last = last;
    this.status = status;

    this.say = function () {
        alert("name: " + this.first + " " + this.last +
              ", status: " + this.status);
    };
}

function run() {

    var proto = new Customer("n/a", "n/a", "pending");
    var prototype = new CustomerPrototype(proto);

    var customer = prototype.clone();
    customer.say();
}
```

Run