The Prototype Pattern

The **Prototype Pattern** specify the kind of objects to create using a prototypical instance, and create new objects by copying this prototype.

Toni Sellarès
Universitat de Girona

Prototype Pattern: Motivation

Use the Prototype Pattern when a client needs to create a set of objects that are alike or differ from each other only in terms of their state and creating an instance of a such object (e.g., using the "new" keyword) is either expensive or complicated.

The Prototype Pattern allows you to make new instances by copying existing instances.

- In Java this typically means using the clone() method or de-serialization when you need deep copies

Key aspect of this pattern:

- Client code can make new instances without knowing which specific class is being instantiated.

Prototype Pattern: Definition

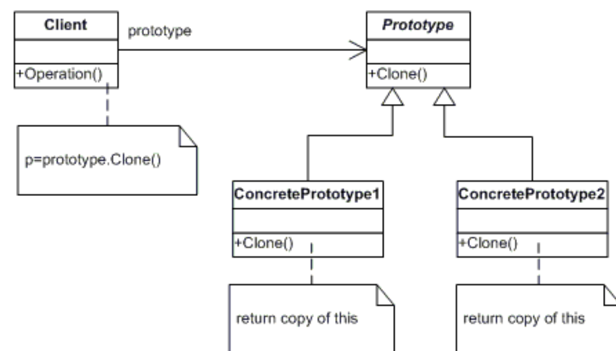
The **Prototype Pattern** specify the kind of objects to create using a prototypical instance, and create new objects by copying this prototype.

Participants

Prototype: declares an interface for cloning itself.

ConcretePrototype: implements an operation for cloning itself.

Client: creates a new object by asking a prototype to clone itself and then making required modifications.



Prototype Pattern: Implementation

To implement the pattern, declare an abstract base class that specifies a pure virtual clone() method.

Any class that needs a "polymorphic constructor" capability derives itself from the abstract base class, and implements the clone() operation.

Clone can be implemented either as a deep copy or a shallow copy:

- In a deep copy, all objects are duplicated,
- In a shallow copy, only the top-level objects are duplicated and the lower levels contain references.

Prototype Pattern: Structural Code

```
/**
 * Test driver for the pattern.
 */

public class Test {
       public static void main( String arg[] ) {
            Client client = new Client();
            Prototype copy = client.operation();
       }
}

/**
 * Declares an interface for cloning itself.
 */

public interface Prototype {
         Prototype copy();
}
```

```
/**
 * Implements an operation for cloning itself.
 */
public class ConcretePrototype1 implements Prototype {
        private String state1 = "Blueprint";
        private Prototype state2;
        public Prototype copy(){
            ConcretePrototype1 duplicate = new ConcretePrototype1();
            duplicate.setState1( new String( state1 ));
                 if( state2 != null ){
                         duplicate.setState2( state2.copy() );
                 }
                 return duplicate;
        }
        void setState1( String state ) {
                 state1 = state;
        }
        void setState2( Prototype state ){
                 state2 = state;
        }
}
```

```
/**
 * Creates a new object by asking a prototype to clone itself.
 */

public class Client {
        public Prototype operation() {
           Prototype prototype = new ConcretePrototype1();
                Prototype copy = prototype.copy();
                return copy;
        }
}
```

Prototype Pattern: Example

This example will create an Address object, which it will then duplicate by calling the object's clone method.

```
public class RunPrototypePattern {
  public static void main(String[] arguments) {
    System.out.println("Creating first address.");
    Address address1 = new Address("8445 Silverado Trail", "Rutherford",
        "CA", "91734");
    System.out.println("First address created.");
    System.out.println("    Hash code = " + address1.hashCode());
    System.out.println(address1);
    System.out.println();
    System.out.println("Creating second address using the clone() method.");
    Address address2 = (Address) address1.copy();
    System.out.println("Second address created.");
    System.out.println("    Hash code = " + address2.hashCode());
    System.out.println(address2);
    System.out.println();
  }
}
```

```java
interface Copyable {
  public Object copy();
}

class Address implements Copyable {
  private String type;
  private String street;
  private String city;
  private String state;
  private String zipCode;
  public static final String EOL_STRING = System
      .getProperty("line.separator");
  public static final String COMMA = ",";
  public static final String HOME = "home";
  public static final String WORK = "work";
  public Address(String initType, String initStreet, String initCity,
      String initState, String initZip) {
    type = initType;
    street = initStreet;
    city = initCity;
    state = initState;
    zipCode = initZip;
  }
```

```java
 public Address(String initStreet, String initCity, String initState,
      String initZip) {
    this(WORK, initStreet, initCity, initState, initZip);
  }
  public Address(String initType) {
    type = initType;
  }
  public Address() {
  }
  public String getType() {
    return type;
  }
  public String getStreet() {
    return street;
  }
  public String getCity() {
    return city;
  }
  public String getState() {
    return state;
  }
 public String getZipCode() {
    return zipCode;
  }
```

```java
  public void setType(String newType) {
    type = newType;
  }
  public void setStreet(String newStreet) {
    street = newStreet;
  }
  public void setCity(String newCity) {
    city = newCity;
  }
  public void setState(String newState) {
    state = newState;
  }
  public void setZipCode(String newZip) {
    zipCode = newZip;
  }
  public Object copy() {
    return new Address(street, city, state, zipCode);
  }
  public String toString() {
    return "\t" + street + COMMA + " " + EOL_STRING + "\t" + city + COMMA
        + " " + state + " " + zipCode;
  }
}
```

Benefits, Uses and Drawbacks

- Benefits:
  - Hides the complexities of making new instances from the client,
  - Provides the option for the client to generate objects whose type is not known,
  - In some circumstances, copying an object can be more efficient than creating a new object.
- Uses:
  - Prototype should be considered when a system must create new objects of many types in a complex class hierarchy.
- Drawbacks:
  - A drawback to using the Prototype is that making a copy of an object can sometimes be complicated.

Abstract Factory ans Protype Patterns may work together.