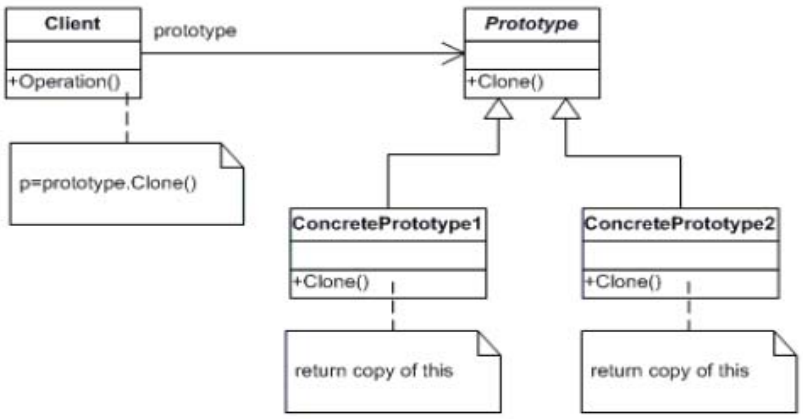
PROTOTYPE PATTERN

Enginyeria del Software I
UdG
Darío Madrid Alonso

Description

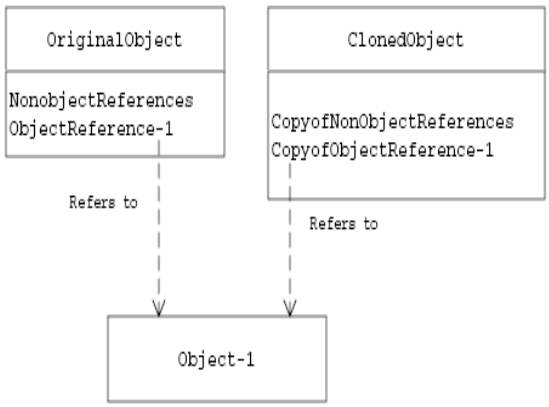
- The Prototype Pattern is used to create new instances by copying existing instances
  - Client makes “news” without knowing what specific class is being instantiated!
  - How? Cloning methods:
    - Shallow Copy
    - Deep Copy
- Motivation: when creating an instance is expensive or complicated

Definition:

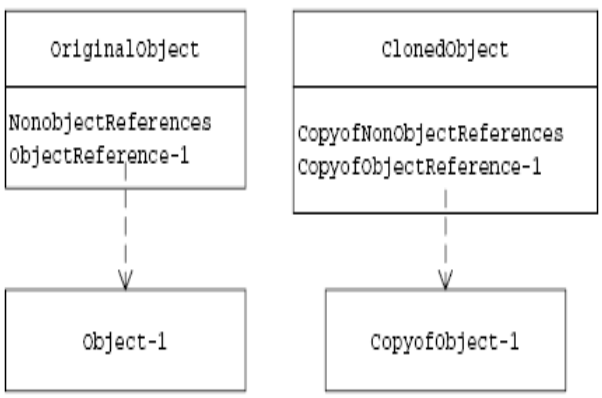


Shallow Copy vs. Deep Copy

Shallow Copy



Deep Copy



Shallow Copy : Implementation

- How?
  - Our Prototype class inherits from “cloneable” interface class
  - Uses “clone” method (java.lang.Object clone())
    - super.clone()
  - Only clones primitives types (integers, strings...)
  - Example:
    - private String name; ➡ Clone ok!
    - private Car car; ➡ Only clones pointer!

Deep Copy : Implementation

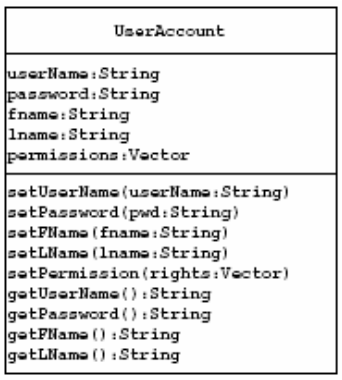
- How?
  - Our Prototype class inherits from “cloneable” interface class
  - Overrides “clone” method (java.lang.Object clone())
  - We've to clone manually all fields or using the constructor if avaiable
  - Example:
    - public Object clone() {
      Person p = new person(name, car.getName());
      return p;
      }

EXAMPLE : User Accounts

- Consider a computer user associated with a user account
- A user account can be part of one or more groups
- Permissions are defined at the group level
- Only 2 groups: Supervisors and AccountRep
- User account stores data(firstname, lastnames,…) and a vector defining permissions

EXAMPLE : First Solution

- Stores the permission for each group in two files (each per group)
- To create a user account:
  - Instantiate the UserAccount class
  - Read Permissions from an appropriate data file
  - Set these permissions in the UserAccount Class
- But…this solution is expensive I/O!!
  - Each new user needs to access the files!



EXAMPLE : Redesign

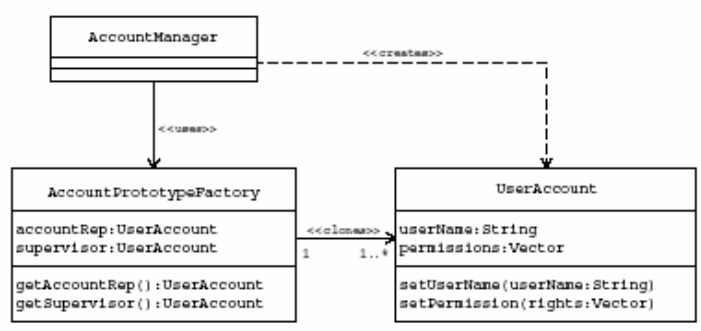
- Designing the UserAccount class to implement the Cloneable interface
- Returning a shallow copy of itself as part of its implementation of the clone method

```
public class UserAccount implements Cloneable {
  private String userName;
  private String password;
  private String fname;
  private String lname;
  private Vector permissions = new Vector();
          …
          …
  public Object clone() {
    //Shallow Copy
    try {
      return super.clone();
    } catch (CloneNotSupportedException e) {
      return null;
    }
           …
           …
}
```

This cloned OK

This not. But ok, because we want in this way.

EXAMPLE : Redesign



- Define a new class, AccountPrototypeFactory, to hold prototypical UserAccounts Objects
- This, returns a copy (clone) of an appropiate UserAccount object

EXAMPLE : Redesign

```
public class AccountPrototypeFactory {
  private UserAccount accountRep;
  private UserAccount supervisor;
  public AccountPrototypeFactory(UserAccount supervisorAccount,
      UserAccount arep) {
    accountRep = arep;
    supervisor = supervisorAccount;
  }
  public UserAccount getAccountRep() {
    return (UserAccount) accountRep.clone();
  }
  public UserAccount getSupervisor() {
    return (UserAccount) supervisor.clone();
  }
}
```

- **Typical client:**
  - Creates two user accounts objects representing Supervisor and AccountRep and Stored in AccountPrototypeFactory as prototype objects
  - Only need one time foreach group to read the permissions from a file
  - After, client invokes methods “get” and receives the cloned objects