Creational Design Pattern

Prototype

Upcode Software
Engineer Team

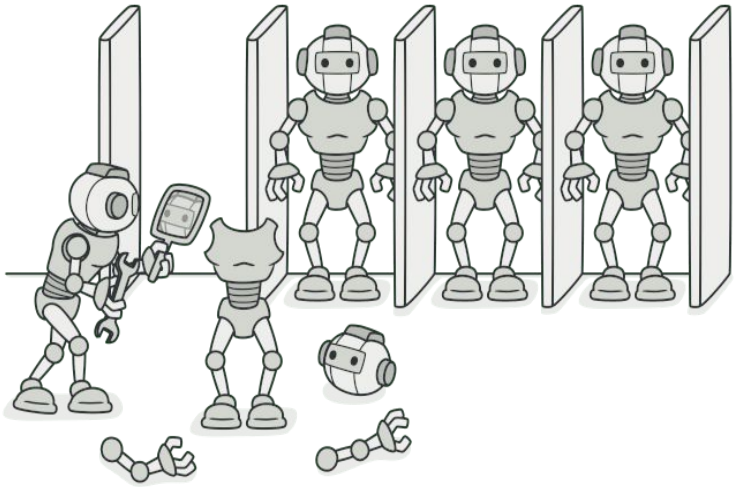
-2023-

CONTENT

1. What is PROTOTYPE ?
2. WHY needs to PROTOTYPE ?
3. HOW to use PROTOTYPE ?
4. WHERE to use PROTOTYPE ?
5. SOURCE code.

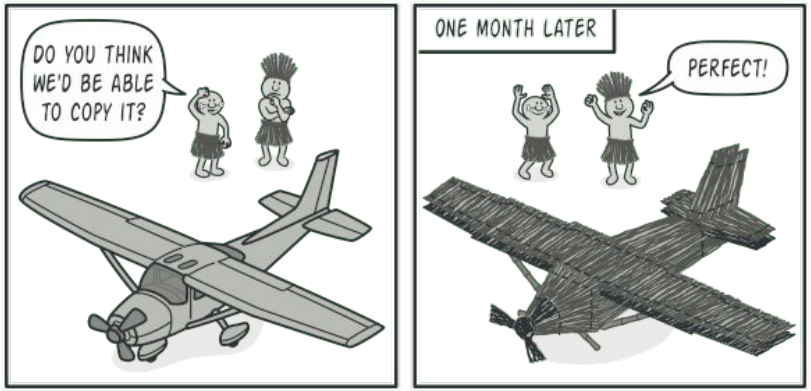
1. What is PROTOTYPE ?

- Prototype is a creational design pattern that lets us copy existing objects without making our code dependent on their classes.
- Also known as: **Cloning Pattern**



2. WHY needs to PROTOTYPE ?

- We want to create an exact copy of some object. But not all objects can be copied that way because some of the object's fields may be private and not visible from outside of the object itself.
- Difficult to instantiate / create new objects because of complex parameters, lengthy instantiation time.



2. WHY needs to PROTOTYPE ?

The main advantages of prototype pattern are as follows:

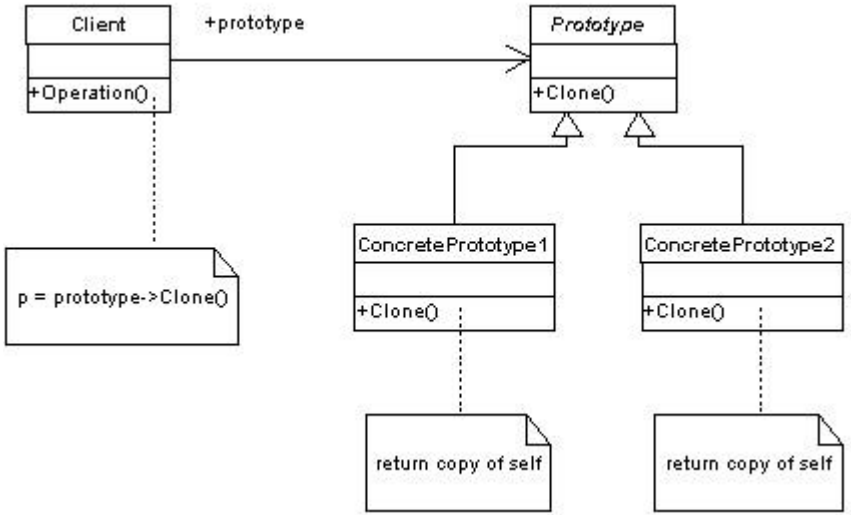
- It reduces the need of sub-classing.
- It hides complexities of creating objects.
- The clients can get new objects without knowing which type of object it will be.
- It lets you add or remove objects at runtime.

3. HOW to use PROTOTYPE ?

- Create an abstract superclass / interface with **clone()** method
- Create subclasses that override that clone() method
- Instantiate one object from each class
- clone that object
- caste that object to the superclass type

3. HOW to use PROTOTYPE ? (2/n)

- **Prototype**: declares an interface for cloning itself.
- **Concrete Prototype**: implements an operation for cloning itself.
- **Client**: creates a new object by asking a prototype to clone itself.

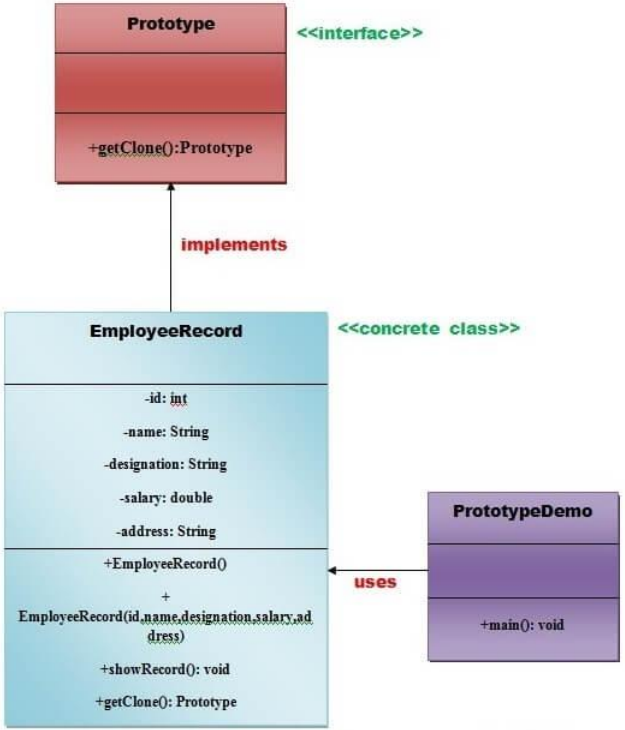


4. WHERE to use PROTOTYPE ?

Usage of Prototype Pattern:

- When the classes are instantiated at runtime.
- When the cost of creating an object is expensive or complicated.
- When creating an instance of a class is time-consuming or complex in some way
- When you want to keep the number of classes in an application minimum.
- Using Prototype design pattern hides concrete product classes from the client.
- Use the Prototype pattern when our code shouldn’t depend on the concrete classes of objects that we need to copy.
- Use the pattern when we want to reduce the number of subclasses that only differ in the way they initialize their respective objects.

5. SOURCE code



5. SOURCE code

```
interface Prototype {

    public Prototype getClone();

}//End of Prototype interface.
```

5. SOURCE code

```
class EmployeeRecord implements Prototype{

  private int id;
  private String name, designation;
  private double salary;
  private String address;

  public EmployeeRecord(){
        System.out.println("   Employee Records of Oracle Corporation ");
        System.out.println("-----------------------------------------------");
        System.out.println("Eid"+"\t"+"Ename"+"\t"+"Edesignation"+"\t"+"Esalary"+"\t\t"+"Eaddress");

}

 public  EmployeeRecord(int id, String name, String designation, double salary, String address) {

      this();
      this.id = id;
      this.name = name;
      this.designation = designation;
      this.salary = salary;
      this.address = address;
   }

  public void showRecord(){

      System.out.println(id+"\t"+name+"\t"+designation+"\t"+salary+"\t"+address);
  }

   @Override
   public Prototype getClone() {

      return new EmployeeRecord(id,name,designation,salary,address);
   }
}//End of EmployeeRecord class.
```

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

class PrototypeDemo{
    public static void main(String[] args) throws IOException {

      BufferedReader br =new BufferedReader(new InputStreamReader(System.in));
      System.out.print("Enter Employee Id: ");
      int eid=Integer.parseInt(br.readLine());
      System.out.print("\n");

      System.out.print("Enter Employee Name: ");
      String ename=br.readLine();
      System.out.print("\n");

      System.out.print("Enter Employee Designation: ");
      String edesignation=br.readLine();
      System.out.print("\n");

      System.out.print("Enter Employee Address: ");
      String eaddress=br.readLine();
      System.out.print("\n");

      System.out.print("Enter Employee Salary: ");
      double esalary= Double.parseDouble(br.readLine());
      System.out.print("\n");

      EmployeeRecord e1=new EmployeeRecord(eid,ename,edesignation,esalary,eaddress);

      e1.showRecord();
      System.out.println("\n");
      EmployeeRecord e2=(EmployeeRecord) e1.getClone();
      e2.showRecord();
    }
}//End of the PrototypeDemo class.
```

5. SOURCE code

1. Prototype Design Pattern (Github)
2. Prototype Design Pattern Computer Types (refactoring.guru)

Summary

- Prototype design pattern is used when the Object creation is a costly affair and requires a lot of time and resources and we have a similar object already existing. Prototype pattern provides a mechanism to copy the original object to a new object and then modify it according to our needs.
- If the object cloning was not provided, we will have to make database call to fetch the data list every time. Then do the manipulations that would have been resource and time consuming.
- That’s all for prototype design pattern.

Reference

1. Dive into Design Pattern (page 123 - 146)
2. Website (www.refactoring.guru)
3. Prototype Design Pattern
4. Website (www.digitalocean.com)
5. Prototype Design Pattern in detail (YouTube)

Thank you!

Presented by **Sanjar Suyunov**

(sanjarsuyunov1304@gmail.com)