# Prototype Pattern in Java

Pankaj Kumar

**Prototype pattern** is one of the Creational Design pattern, so it provides a mechanism of object creation. Prototype pattern is used when the Object creation is a costly affair and requires a lot of time and resources and you have a similar object already existing. So this pattern provides a mechanism to copy the original object to a new object and then modify it according to our needs. This pattern uses java cloning to copy the object.

It would be easy to understand this pattern with an example, suppose we have an Object that loads data from database. Now we need to modify this data in our program multiple times, so its not a good idea to create the Object using new keyword and load all the data again from database. So the better approach is to clone the existing object into a new object and then do the data manipulation.

Prototype design pattern mandates that the Object which you are copying should provide the copying feature. It should not be done by any other class. However whether to use shallow or deep copy of the Object properties depends on the requirements and its a design decision.

Here is a sample program showing implementation of Prototype pattern.

|  |
| --- |
| **package** org.xman.dp.creational.prototype;  **import** java.util.ArrayList;  **import** java.util.List;  **public** **class** Employees **implements** Cloneable {  **private** List<String> empList;  **public** **Employees**() {  empList = **new** ArrayList<String>();  }  **public** **Employees**(List<String> list) {  **this**.empList = list;  }  **public** **void** **loadData**() {  // read all employees from database and put into the list  empList.add("Pankaj");  empList.add("Raj");  empList.add("David");  empList.add("Lisa");  }  **public** List<String> **getEmpList**() {  **return** empList;  }  @Override  **public** Object **clone**() **throws** CloneNotSupportedException {  List<String> temp = **new** ArrayList<String>();  **for** (String s : **this**.getEmpList()) {  temp.add(s);  }  **return** **new** Employees(temp);  }  } |

Notice that the clone method is overridden to provide a deep copy of the employees list.

Here is the test program that will show the benefit of prototype pattern usage.

|  |
| --- |
| **package** org.xman.dp.creational.prototype;  **import** java.util.List;  **public** **class** PrototypePatternTest {  **public** **static** **void** **main**(String[] args) **throws** CloneNotSupportedException {  Employees emps = **new** Employees();  emps.loadData();  // Use the clone method to get the Employee object  Employees empsNew = (Employees) emps.clone();  Employees empsNew1 = (Employees) emps.clone();  List<String> list = empsNew.getEmpList();  list.add("John");  List<String> list1 = empsNew1.getEmpList();  list1.remove("Pankaj");  System.out.println("emps List: " + emps.getEmpList());  System.out.println("empsNew List: " + list);  System.out.println("empsNew1 List: " + list1);  }  } |

Output of the above program is:

emps HashMap: [Pankaj, Raj, David, Lisa]

empsNew HashMap: [Pankaj, Raj, David, Lisa, John]

empsNew1 HashMap: [Raj, David, Lisa]

If the object cloning was not provided, every time we need to make database call to fetch the employee list and then do the manipulations that would have been resource and time consuming.