Prototype

Resources

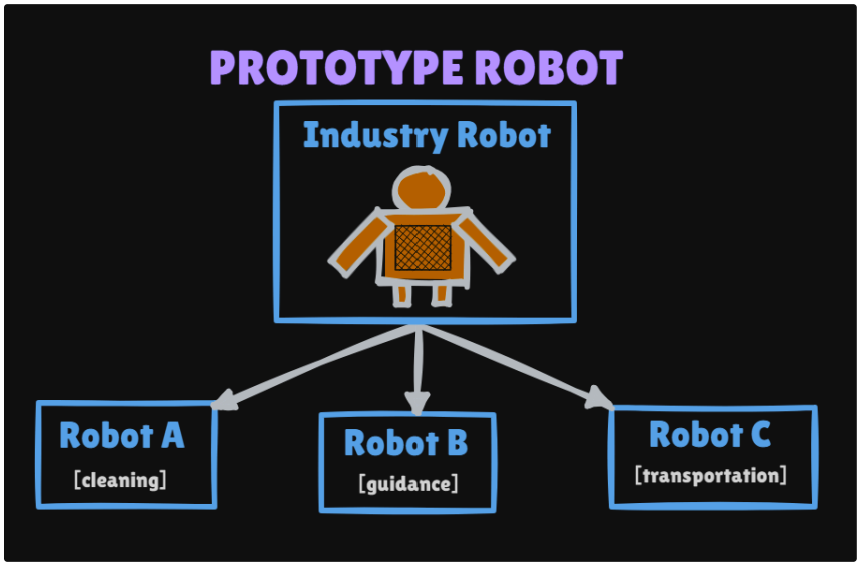
- Video → 27. All Creational Design Patterns | Prototype, Singleton, Factory, AbstractFactory, Builder Pattern

Definition

> *The Prototype design pattern is a creational pattern that allows you to **create new objects by cloning existing instances** rather than creating them from scratch.*

This is particularly useful when object creation is **expensive** or when you want to avoid the complexities involved in creating objects using constructors.

Example: Robot



Creating each type of task-oriented robot from scratch is expensive. Instead, we clone the basic model of an industrial robot (the prototype) and customize it for specific tasks.

The Problem: Student Example

```
1
2  // Concrete Class - Whose clone is to be created
3  public class Student {
4
```

```
    int id;
    String name;
    String branch;
    private int rollNo;

    Student() {

    }

    Student(int id, String name, String branch, int rollNo) {
        this.id = id;
        this.name = name;
        this.branch = branch;
        this.rollNo = rollNo;
    }

    public void printDetails() {
        System.out.println("=== Student Details ===");
        System.out.print(this + ": ");
        System.out.println("Id: " + id + ", Name: " + name + ",
Branch: " + branch + ", Roll No: " + rollNo);
    }
}

// Client
public class DemoProblem {
    public static void main(String[] args) {
        Student studentOrg = new Student(1, "Aman", "CSE", 123);
        studentOrg.printDetails();
        // create a clone of the student object
        Student studentClone = new Student();
        studentClone.id = studentOrg.id;
        studentClone.name = studentOrg.name;
        studentClone.branch = studentOrg.branch;
        // studentClone.rollNo = studentOrg.rollNo; // Compilation
error
    }
}
```

Drawbacks

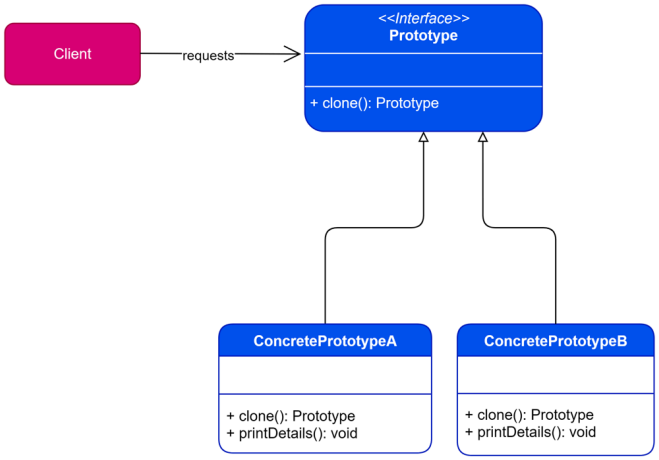
- Can't access `private` fields outside the class during cloning. Ex: `rollNo` in the `Student` class.
- It is not possible to be selective about the properties and behavior when cloning.

Solution: The Prototype Design Pattern

Instead of the client taking the responsibility of cloning the object, the cloning responsibility is assigned to the actual concrete class implementing the prototype interface to clone itself.

The prototype interface ensures consistent implementation of cloning objects with features common to all subtypes.

Class Diagram



Implementation

```java
// Prototype interface
public interface StudentPrototype {
    StudentPrototype clone();
}

// Concrete prototype class
public class Student implements StudentPrototype {

    int id;
    String name;
    String branch;
    boolean inHighSchool;
    private int rollNo;

    Student() {

    }

    // Clone constructor for efficient copying
    Student(int id, String name, String branch, int rollNo) {
        this.id = id;
        this.name = name;
        this.branch = branch;
        this.rollNo = rollNo;
    }

    // setter method
    public void setInHighSchool(boolean inHighSchool) {
        this.inHighSchool = inHighSchool;
    }

    @Override
    public StudentPrototype clone() {
        return new Student(id, name, branch, rollNo);
    }

    public void printDetails() {
        System.out.println("=== Student Details ===");
        System.out.print(this + ": ");
        System.out.println("Id: " + id + ", Name: " + name + ",
Branch: " + branch + ", Roll No: " + rollNo + ", In High School: " +
inHighSchool);
    }
}

// Client
public class DemoSolution {
    public static void main(String[] args) {
```

```
        // Create initial prototypes (expensive operations)
        Student student = new Student(5, "Rita", "CSE", 224);
        student.printDetails();
        // Clone objects (fast operations)
        Student studentClone = (Student) student.clone();
        studentClone.setInHighSchool(true);
        studentClone.printDetails();
        System.out.println("Same object? " + (student ==
studentClone)); // false
    }
}
```