

1. **Types of Patterns:**
   1. **Creational Patterns**: Focus on flexible, decoupled object creation mechanisms, such as the Singleton, Factory, and Builder patterns.
   2. **Structural Patterns**: Deal with the efficient composition of classes and objects to form larger structures, including the Adapter, Decorator, and Composite patterns.
   3. **Behavioral Patterns**: Focus on managing the interactions and behavior between objects, such as the Observer, Strategy, and Command patterns.

public class Singleton {

private static Singleton instance;

// Private constructor to prevent instantiation

private Singleton() {

// Private constructor

}

// Public method to provide access to the instance

public static Singleton getInstance() {

if (instance == null) {

instance = new Singleton();

}

return instance;

}

// Method to demonstrate functionality (optional)

public void showMessage() {

System.out.println("Singleton instance is working!");

}

public static void main(String[] args) {

// Get the only instance of Singleton

Singleton singleton = Singleton.getInstance();

// Call a method on the Singleton instance

singleton.showMessage();

// Verifying that only one instance is created

Singleton anotherInstance = Singleton.getInstance();

System.out.println("Are both instances equal? " + (singleton == anotherInstance));

}

}}

Prototype   
  
  
public class Vehicle implements Cloneable {

private String type;

public Vehicle(String type) {

this.type = type;

}

public String getType() {

return type;

}

public void setType(String type) {

this.type = type;

}

@Override

public Object clone() {

try {

return super.clone();

} catch (CloneNotSupportedException e) {

throw new AssertionError(); // Shouldn't happen since we implement Cloneable

}

}

@Override

public String toString() {

return "Vehicle{type='" + type + "'}";

}

public static void main(String[] args) {

Vehicle vehicle1 = new Vehicle("Car");

try {

Vehicle vehicle2 = (Vehicle) vehicle1.clone();

System.out.println(vehicle1); // Vehicle{type='Car'}

System.out.println(vehicle2); // Vehicle{type='Car'}

// Modify the clone

vehicle2.setType("Bike");

System.out.println(vehicle1); // Vehicle{type='Car'}

System.out.println(vehicle2); // Vehicle{type='Bike'}

} catch (CloneNotSupportedException e) {

e.printStackTrace();

}

}

}

