The Prototype Pattern is a creational design pattern that focuses on creating new objects by cloning existing ones.

It involves creating a prototype object, which acts as a blueprint, and then copying or cloning that prototype to create new objects.

The Prototype Pattern is useful when creating new objects is costly or complex, and it's more efficient to clone existing objects instead.

*interface Shape extends Cloneable* {
 *void* draw();
 *Shape* clone();
}

*class* Circle *implements Shape* {
 *public void* draw() {
 System.*out*.println("Drawing a circle");
 }

 @Override
 *public* Circle clone() {
 *try* {
 *return* (Circle) *super*.clone();
 } *catch* (CloneNotSupportedException e) {
 *throw new* RuntimeException("Cloning not supported", e);
 }
 }
}

*class* Rectangle *implements Shape* {
 *public void* draw() {
 System.*out*.println("Drawing a rectangle");
 }

 @Override
 *public* Rectangle clone() {
 *try* {
 *return* (Rectangle) *super*.clone();
 } *catch* (CloneNotSupportedException e) {
 *throw new* RuntimeException("Cloning not supported", e);
 }
 }
}

*public class* Prototype2 {
 *public static void* main(String[] args) {
 *Shape* circlePrototype = *new* Circle();
 *Shape* rectanglePrototype = *new* Rectangle();

 *Shape* circle1 = circlePrototype.clone();
 *Shape* circle2 = circlePrototype.clone();
 *Shape* rectangle1 = rectanglePrototype.clone();
 *Shape* rectangle2 = rectanglePrototype.clone();
 circle1.draw();
 circle2.draw();
 rectangle1.draw();
 rectangle2.draw();

 }
}

Efficiency: Cloning existing objects can be more efficient than creating new instances from scratch,

especially if object creation involves complex initialization or resource allocation.

Resource Sharing: Cloned objects can share resources or data with the prototype,

reducing memory consumption and improving performance.

Dynamic Configuration: You can create new objects by cloning and then modifying the cloned objects,

allowing for dynamic configurations.

Simplified Object Creation: The client code doesn't need to know the specific class of the prototype to create new objects.

The Prototype Pattern is particularly useful when:

Object creation is costly or complex.

You want to create new objects by copying existing ones.

You want to share resources or data between objects.

You need to create objects with varying configurations.