Patrón Prototipo

Jose Andres Matarrita Miranda C04668

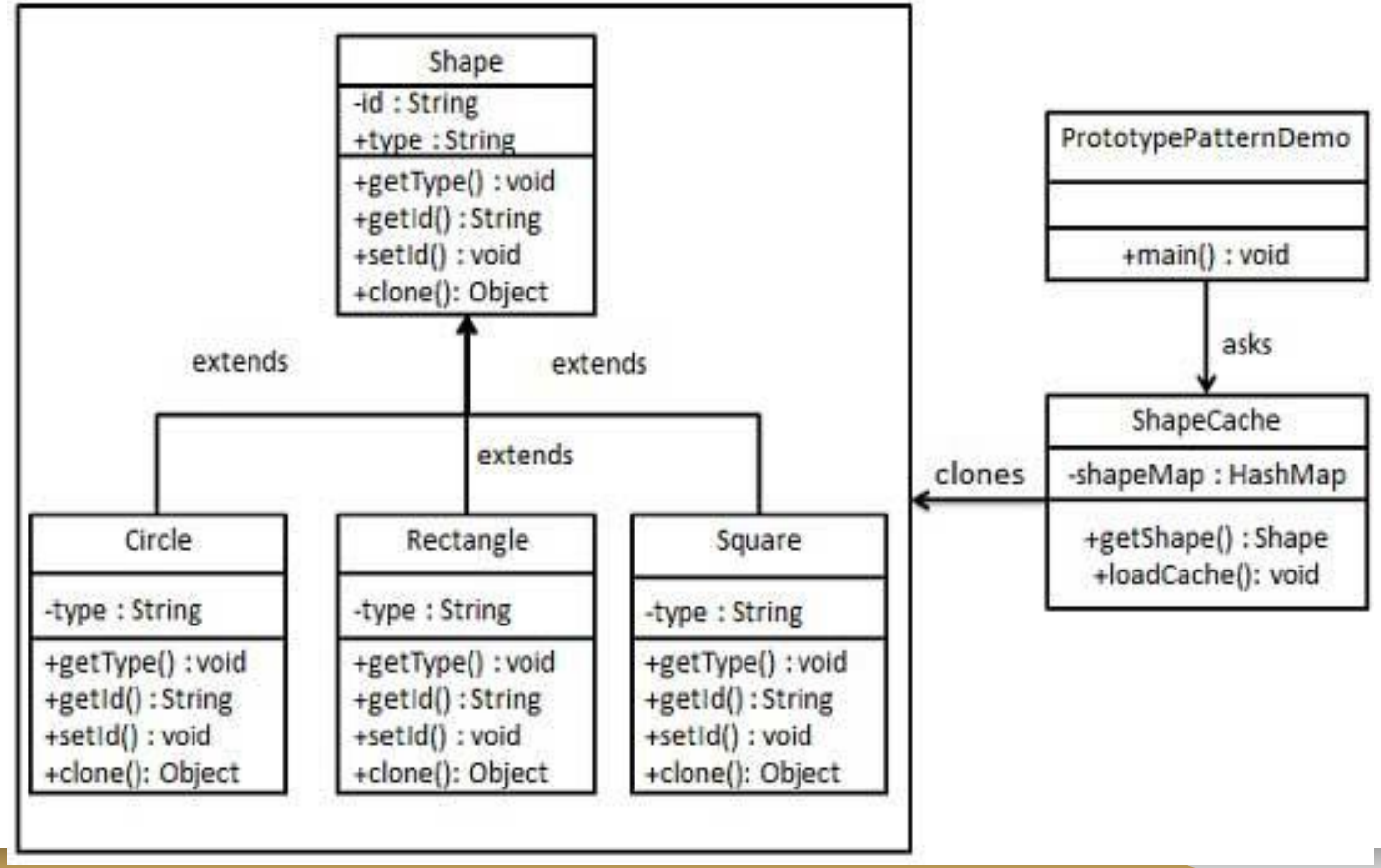
Definición:

- Patrón de diseño creacional
- Crea objetos a partir de un prototipo en lugar de crearlos desde cero
- Utiliza una copia del objeto original para crear nuevos objetos

Problema:

- Creación de objetos es costosa
- Creación de objetos puede ser complejo
- Permitir la creación de objetos a partir de una instancia inicial

Solución que se plantea



```
public:
    virtual ~Shape() {}
    virtual void draw() = 0;
    std::string getType() {
        return type;
    }
    std::string getId() {
        return id;
    }
    void setId(std::string id) {
        this->id = id;
    }
    std::string getColor() { // Getter para el color
        return color;
    }
    void setColor(std::string color) { // Setter para el color
        this->color = color;
    }
    virtual Shape* clone() = 0;
};

// También se tendría que actualizar las clases derivadas Rectangle, Square, y
```

```cpp
class Rectangle : public Shape {
public:
    Rectangle() {
        type = "Rectangle";
    }
    void draw() {
        std::cout << "Inside Rectangle::draw() method." << std::endl;
    }
    Shape* clone() {
        return new Rectangle(*this);
    }
};

class Square : public Shape {
public:
    Square() {
        type = "Square";
    }
    void draw() {
        std::cout << "Inside Square::draw() method." << std::endl;
    }
    Shape* clone() {
        return new Square(*this);
    }
};
```

```
static void loadCache() {
    auto circle = new Circle();
    circle->setId("1");
    circle->setColor("red"); // Establecer el color del círculo
    cache[circle->getId()] = circle;

    auto rectangle = new Rectangle();
    rectangle->setId("2");
    rectangle->setColor("blue"); // Establecer el color del rectángulo
    cache[rectangle->getId()] = rectangle;

    auto square = new Square();
    square->setId("3");
    square->setColor("green"); // Establecer el color del cuadrado
    cache[square->getId()] = square;
}
```

```cpp
int main() {
    ShapeCache::loadCache();

    auto clonedShape1 = ShapeCache::getShape("1");
    std::cout << "Shape : " << clonedShape1->getType() << ", Color: " << clonedShape1->getColor() << std::endl;

    auto clonedShape2 = ShapeCache::getShape("2");
    std::cout << "Shape : " << clonedShape2->getType() << ", Color: " << clonedShape2->getColor() << std::endl;

    auto clonedShape3 = ShapeCache::getShape("3");
    std::cout << "Shape : " << clonedShape3->getType() << ", Color: " << clonedShape3->getColor() << std::endl;

    return 0;
}
```

Casos es recomendado utilizar este patrón.

- Objetos complejos
- Creación de objetos costosa
- Compartir estructura básica entre objetos
- Menor costo de creación de objetos
- Clonación de objetos
- Flexibilidad en la creación de objetos
- Reducción de la sobrecarga de memoria
- Aceleración de la creación de objetos
- Aplicaciones que requieren la creación frecuente de objetos

Ventajas:

- Reducción de costos y complejidad en la creación de objetos.
- Mayor eficiencia en la creación de objetos complejos.
- Flexibilidad en la creación de nuevos objetos a partir de prototipos existentes.

Desventajas:

- Puede requerir una configuración adicional para establecer los prototipos.
- Puede haber una mayor complejidad en la gestión de los objetos prototipos.
- El objeto prototipo inicial puede necesitar ser cuidadosamente diseñado para adaptarse a todas las posibles variaciones de objetos que se pueden crear a partir de él.