# Notepad/ Text Editor

**Design a simple text editor with support for:**

* Text insertion
* Cursor movement
* Deletion
* Undo/Redo

**Components Implemented**

|  |  |
| --- | --- |
| Component | Description |
| Editor | Central class maintaining buffer, cursor, undo/redo stacks |
| Command | Abstract base class for all commands (Insert, Delete, MoveCursor) |
| InsertCommand | Inserts text at the current cursor position |
| DeleteCommand | Deletes characters before the cursor (backspace-style) |
| MoveCursorCommand | Moves the cursor by a given offset |

**Design Patterns Used**

|  |  |
| --- | --- |
| Pattern | Where/How it’s Used |
| Command Pattern | Each action (Insert, Delete, MoveCursor) is implemented as a Command class |
| Invoker | The Editor acts as the invoker — it calls command.execute() and maintains history |
| Undo/Redo Stack | Enables reversing actions by calling command.undo() |
| *(Optional)* Observer | Could be added later to notify UI components on state change |

**SOLID Principles Applied**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Key Takeaways**

**A black background with white text

AI-generated content may be incorrect.**

**A computer screen with text and images

AI-generated content may be incorrect.A screen shot of a computer program

AI-generated content may be incorrect.**

**A screen shot of a computer code

AI-generated content may be incorrect.A screen shot of a computer

AI-generated content may be incorrect.A computer screen shot of a program

AI-generated content may be incorrect.A screenshot of a computer program

AI-generated content may be incorrect.**

## A screenshot of a computer program AI-generated content may be incorrect.

## **If You Need More Methods?**

You could have more methods, like redo(), validate(), or get\_state(), **but they are not essential** to the pattern. You’d only add them if your system explicitly requires those features — e.g.:

* validate() – if you want to prevent invalid commands
* redo() – often redo() is just calling execute() again
* log() – for auditing/logging purposes

# LRU Cache

## What It Is: An in-memory cache that:

* Stores up to **N items** (capacity)
* When full, removes the **Least Recently Used** item
* Provides **O(1)** access to:
  + get(key) → returns value
  + put(key, value) → insert/update key-value pair

## Core Requirements

|  |  |
| --- | --- |
| Feature | Requirement |
| get(key) | Return value if exists, else -1. Mark as most recently used. |
| put(key, val) | Insert or update key-value. If full, evict LRU item. |

|  |  |
| --- | --- |
| Operation | Action |
| get(key) | Move node to head (most recently used) |
| put(key, value) | Insert/update node, evict tail if over capacity |

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.A screenshot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

1. Parking Lot System

A screenshot of a computer

AI-generated content may be incorrect.

## Problem Statement

**Design a parking lot system that can:**

1. **Park a vehicle**
2. **Unpark a vehicle**
3. **Track available spots**
4. Handle **different vehicle types**: car, bike, truck, etc.
5. Handle **multiple floors** (for scalability)
6. Handle **Pricing** by **hour** and **type** of vehicle

**SOLID Principles**

A screenshot of a black screen

AI-generated content may be incorrect.

**Design Patterns  
  
A screenshot of a black screen

AI-generated content may be incorrect.**

## Summary in Plain English

* Each class **does one job** (SRP).
* We’ve structured the system so that it’s **easy to extend** (new vehicle types, new pricing, new floors) without touching existing code (OCP).
* If tomorrow you want to introduce **EV spots**, **QR payment**, or **multi-tier pricing**, it fits right in.
* PricingStrategy is a perfect use of the **Strategy pattern**, showing how logic can be modular and swappable.
* The ParkingLot is **cleanly separated from implementation details** like how pricing is calculated (DIP).