1. LRU Cache Implementation

Problem Statement

Design and implement a data structure for a Least Recently Used (LRU) cache. It should support the following operations: `get` and `put`.

Constraints

- The number of get and put operations will be in the range [1, 10^5].
- The capacity of the cache is between 1 and 10^5.

Solution

We can use a combination of a doubly linked list and a hash map to implement an LRU cache. The doubly linked list will help us in maintaining the order of elements, and the hash map will provide O(1) access to elements.

java

```
import java.util.HashMap;

class LRUCache {
    private class Node {
        int key, value;
        Node prev, next;
        Node(int key, int value) {
            this.key = key;
            this.value = value;
        }
    }

    private final int capacity;
    private final HashMap<Integer, Node> map;
    private final Node head, tail;

    public LRUCache(int capacity) {
        this.capacity = capacity;
        this.map = new HashMap<>(capacity);
        head = new Node(0, 0);
```

```
        tail = new Node(0, 0);
        head.next = tail;
        tail.prev = head;
    }

    public int get(int key) {
        if (!map.containsKey(key)) {
            return -1;
        }
        Node node = map.get(key);
        remove(node);
        insert(node);
        return node.value;
    }

    public void put(int key, int value) {
        if (map.containsKey(key)) {
            remove(map.get(key));
        } else if (map.size() == capacity) {
            remove(tail.prev);
        }
        insert(new Node(key, value));
    }

    private void remove(Node node) {
        map.remove(node.key);
        node.prev.next = node.next;
        node.next.prev = node.prev;
    }

    private void insert(Node node) {
        map.put(node.key, node);
        node.next = head.next;
        node.prev = head;
        head.next.prev = node;
        head.next = node;
    }
}
```

2. ConcurrentModificationException Demonstration

Problem Statement

Write a Java program that demonstrates the `ConcurrentModificationException`. Explain why the exception is thrown and how to handle it properly.

Solution

The `ConcurrentModificationException` is thrown when a collection is modified while iterating over it using an iterator. This can be demonstrated and handled as follows:

java

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class ConcurrentModificationDemo {
    public static void main(String[] args) {
        List<Integer> list = new ArrayList<>();
        list.add(1);
        list.add(2);
        list.add(3);

        try {
            for (Integer value : list) {
                if (value == 2) {
                    list.remove(value);
ConcurrentModificationException
                }
            }
        } catch (ConcurrentModificationException e) {
            System.out.println("Caught
ConcurrentModificationException: " + e.getMessage());
        }

        // Correct way to handle modification
        Iterator<Integer> iterator = list.iterator();
        while (iterator.hasNext()) {
```

```
            if (iterator.next() == 2) {
                iterator.remove();
                ConcurrentModificationException
            }
        }

        System.out.println("Modified list: " + list);
    }
}
```

3. Custom Annotation @LogExecutionTime

Problem Statement

Create a custom annotation `@LogExecutionTime` to log the execution time of annotated methods. Implement an annotation processor to handle this annotation.

Solution

To create a custom annotation and an aspect to log execution time, we can use Spring AOP.

Step 1: Create the annotation
java

```
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface LogExecutionTime {
}
```

Step 2: Create the aspect
java

```
import org.aspectj.lang.ProceedingJoinPoint;
```

```
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.springframework.stereotype.Component;

@Aspect
@Component
public class LogExecutionTimeAspect {

    @Around("@annotation(LogExecutionTime)")
    public Object logExecutionTime(ProceedingJoinPoint joinPoint)
throws Throwable {
        long start = System.currentTimeMillis();

        Object proceed = joinPoint.proceed();

        long executionTime = System.currentTimeMillis() - start;

        System.out.println(joinPoint.getSignature() + " executed in "
+ executionTime + "ms");
        return proceed;
    }
}
```

Step 3: Enable aspect in Spring Boot application

java

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.EnableAspectJAutoProxy;

@SpringBootApplication
@EnableAspectJAutoProxy
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

4. Serialize and Deserialize a Binary Tree

Problem Statement

Design an algorithm to serialize and deserialize a binary tree.

Solution

We can use a simple pre-order traversal for serialization and deserialization.

java

```
import java.util.*;

public class Codec {
    private static final String SPLITTER = ",";
    private static final String NULL_NODE = "X";


    public String serialize(TreeNode root) {
        StringBuilder sb = new StringBuilder();
        serializeHelper(root, sb);
        return sb.toString();
    }

    private void serializeHelper(TreeNode root, StringBuilder sb) {
        if (root == null) {
            sb.append(NULL_NODE).append(SPLITTER);
        } else {
            sb.append(root.val).append(SPLITTER);
            serializeHelper(root.left, sb);
            serializeHelper(root.right, sb);
        }
    }

       public TreeNode deserialize(String data) {
        Queue<String> nodes = new
LinkedList<>(Arrays.asList(data.split(SPLITTER)));
        return deserializeHelper(nodes);
    }
```

```
    private TreeNode deserializeHelper(Queue<String> nodes) {
        String val = nodes.poll();
        if (val.equals(NULL_NODE)) {
            return null;
        }
        TreeNode node = new TreeNode(Integer.parseInt(val));
        node.left = deserializeHelper(nodes);
        node.right = deserializeHelper(nodes);
        return node;
    }

        public static class TreeNode {
        int val;
        TreeNode left;
        TreeNode right;
        TreeNode(int val) { this.val = val; }
    }
}
```

5. Implement a Trie

Problem Statement

Implement a trie with `insert`, `search`, and `startsWith` methods.

Solution

java

```
public class Trie {
    private TrieNode root;

    public Trie() {
        root = new TrieNode();
    }

        public void insert(String word) {
        TrieNode node = root;
```

```
    for (char c : word.toCharArray()) {
        if (!node.containsKey(c)) {
            node.put(c, new TrieNode());
        }
        node = node.get(c);
    }
    node.setEnd();
}


public boolean search(String word) {
    TrieNode node = searchPrefix(word);
    return node != null && node.isEnd();
}


public boolean startsWith(String prefix) {
    return searchPrefix(prefix) != null;
}

private TrieNode searchPrefix(String word) {
    TrieNode node = root;
    for (char c : word.toCharArray()) {
        if (node.containsKey(c)) {
            node = node.get(c);
        } else {
            return null;
        }
    }
    return node;
}

class TrieNode {
    private final int R = 26;
    private TrieNode[] links;
    private boolean isEnd;

    public TrieNode() {
```

```
            links = new TrieNode[R];
        }

        public boolean containsKey(char ch) {
            return links[ch - 'a'] != null;
        }

        public TrieNode get(char ch) {
            return links[ch - 'a'];
        }

        public void put(char ch, TrieNode node) {
            links[ch - 'a'] = node;
        }

        public void setEnd() {
            isEnd = true;
        }

        public boolean isEnd() {
            return isEnd;
        }
    }
}
```

6. Valid Parentheses

Problem Statement

Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

Solution

java

```
import java.util.Stack;

public class ValidParentheses {
```

```
    public boolean isValid(String s) {
        Stack<Character> stack = new Stack<>();
        for (char c : s.toCharArray()) {
            if (c == '(') {
                stack.push(')');
            } else if (c == '{') {
                stack.push('}');
            } else if (c == '[') {
                stack.push(']');
            } else if (stack.isEmpty() || stack.pop() != c) {
                return false;
            }
        }
        return stack.isEmpty();
    }
}
```