Solution to Problem 3.4.2

We are tasked with implementing an alternate version of the SeparateChainingHashST class that directly uses the linked list code from SequentialSearchST. Below is the implementation:

Linked-List Symbol Table (SequentialSearchST)

The SequentialSearchST class provides a simple linked list-based implementation of a symbol table.

```java
// Linked-list-based symbol table
class SequentialSearchST<Key, Value> {
    private Node first;

    private class Node {
        Key key;
        Value val;
        Node next;

        public Node(Key key, Value val, Node next) {
            this.key = key;
            this.val = val;
            this.next = next;
        }
    }

    public Value get(Key key) {
        for (Node x = first; x != null; x = x.next) {
            if (key.equals(x.key)) return x.val; // Found key
        }
        return null; // Key not found
    }

    public void put(Key key, Value val) {
        for (Node x = first; x != null; x = x.next) {
            if (key.equals(x.key)) {
                x.val = val; // Update value
                return;
            }
        }
        first = new Node(key, val, first); // Insert new node
    }

    public void delete(Key key) {
        first = delete(first, key);
    }

    private Node delete(Node x, Key key) {
        if (x == null) return null;
        if (key.equals(x.key)) return x.next; // Remove node
        x.next = delete(x.next, key);
        return x;
    }
}
```

Hash Table with Separate Chaining (SeparateChainingHashST)

The SeparateChainingHashST class uses an array of SequentialSearchST objects to handle collisions.

```java
// Hash table with separate chaining
public class SeparateChainingHashST<Key, Value> {
    private static final int DEFAULT_SIZE = 4; // Default table size
    private int m; // Number of chains
    private SequentialSearchST<Key, Value>[] chains;

    public SeparateChainingHashST() {
        this(DEFAULT_SIZE);
    }

    public SeparateChainingHashST(int m) {
        this.m = m;
        chains = (SequentialSearchST<Key, Value>[]) new SequentialSearchST[m];
        for (int i = 0; i < m; i++) {
            chains[i] = new SequentialSearchST<>();
        }
    }

    private int hash(Key key) {
        return (key.hashCode() & 0x7fffffff) % m; // Hash function
    }

    public Value get(Key key) {
        int i = hash(key);
        return chains[i].get(key);
    }

    public void put(Key key, Value val) {
        int i = hash(key);
        chains[i].put(key, val);
    }

    public void delete(Key key) {
        int i = hash(key);
        chains[i].delete(key);
    }
}
```

Usage Example

Below is an example demonstrating how to use the SeparateChainingHashST class.

```java
public class Main {
    public static void main(String[] args) {
        SeparateChainingHashST<String, Integer> hashTable = new SeparateChainingHashST<>(5);

        // Insert key-value pairs
        hashTable.put("E", 1);
        hashTable.put("A", 2);
        hashTable.put("S", 3);
        hashTable.put("Y", 4);

        // Retrieve values
        System.out.println("Value of E: " + hashTable.get("E")); // Output: 1
        System.out.println("Value of S: " + hashTable.get("S")); // Output: 3

        // Delete a key
        hashTable.delete("A");
        System.out.println("Value of A: " + hashTable.get("A")); // Output: null
    }
}
```

Explanation

- The SequentialSearchST class implements a symbol table using a linked list. - The SeparateChainingHashST class uses an array of SequentialSearchST objects, each representing a chain for handling collisions. - This implementation ensures efficient operations like put, get, and delete, leveraging the modular design of SequentialSearchST.