Nayankumar Dhome nayankumardhome@gmail.com

Map Interface – The Power of Key-Value Pairing

Let's Swipe Right →

What is a Map?

- A Map in Java is a collection that maps unique keys to values.
- Keys are unique, but values can be duplicated.
- Null keys and values are supported in some implementations.

Nayankumar Dhome nayankumardhome@gmail.com

Key Features of Map

1. Key-Value Pairing: Allows retrieval of values based on unique keys.
2. No Duplicate Keys: Each key maps to at most one value.
3. Efficient Lookups: Designed for fast data retrieval based on keys.
4. Custom Implementations: Offers different trade-offs between speed, concurrency, and ordering.

Let's Swipe Right →

Nayankumar Dhome nayankumardhome@gmail.com

Key Methods in the Map Interface

- **put(K key, V value): A**ssociates the specified value with the specified key.
- **get(Object key):** Returns the value to which the key is mapped, or null if no mapping exists.
- **remove(Object key):** Removes the mapping for a key if it exists.
- **containsKey(Object key):** Checks if the map contains the specified key.
- **containsValue(Object value):** Checks if the map contains the specified value.
- **keySet():** Returns a set of all keys.
- **values():** Returns a collection of all values.
- **entrySet():** Returns a set of all key-value mappings.

Let's Swipe Right

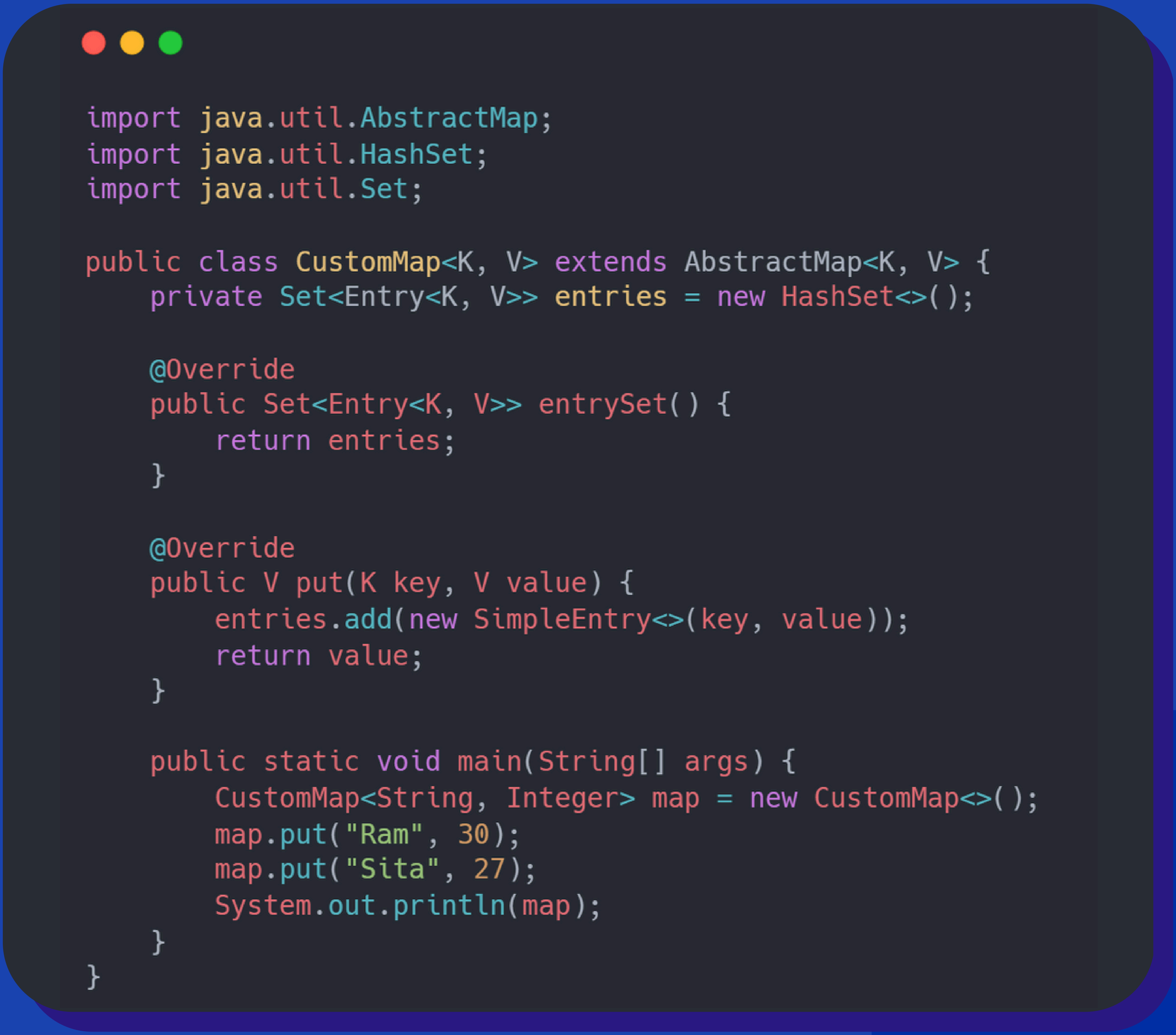
Common Implementations /extends of Map

AbstractMap (Abstract Class)

- **Purpose:** Provides a skeletal implementation of the Map interface to minimize effort required to implement a map.
- **Use Case:** Extend this class to create custom map implementations.
- **Example:** HashMap, TreeMap, and other maps extend this class.

Let's Swipe Right →

AbstractMap (Abstract Class)



```java
import java.util.AbstractMap;
import java.util.HashSet;
import java.util.Set;

public class CustomMap<K, V> extends AbstractMap<K, V> {
    private Set<Entry<K, V>> entries = new HashSet<>();

    @Override
    public Set<Entry<K, V>> entrySet() {
        return entries;
    }

    @Override
    public V put(K key, V value) {
        entries.add(new SimpleEntry<>(key, value));
        return value;
    }

    public static void main(String[] args) {
        CustomMap<String, Integer> map = new CustomMap<>();
        map.put("Ram", 30);
        map.put("Sita", 27);
        System.out.println(map);
    }
}
```

Let's Swipe Right →

Nayankumar Dhome nayankumardhome@gmail.com

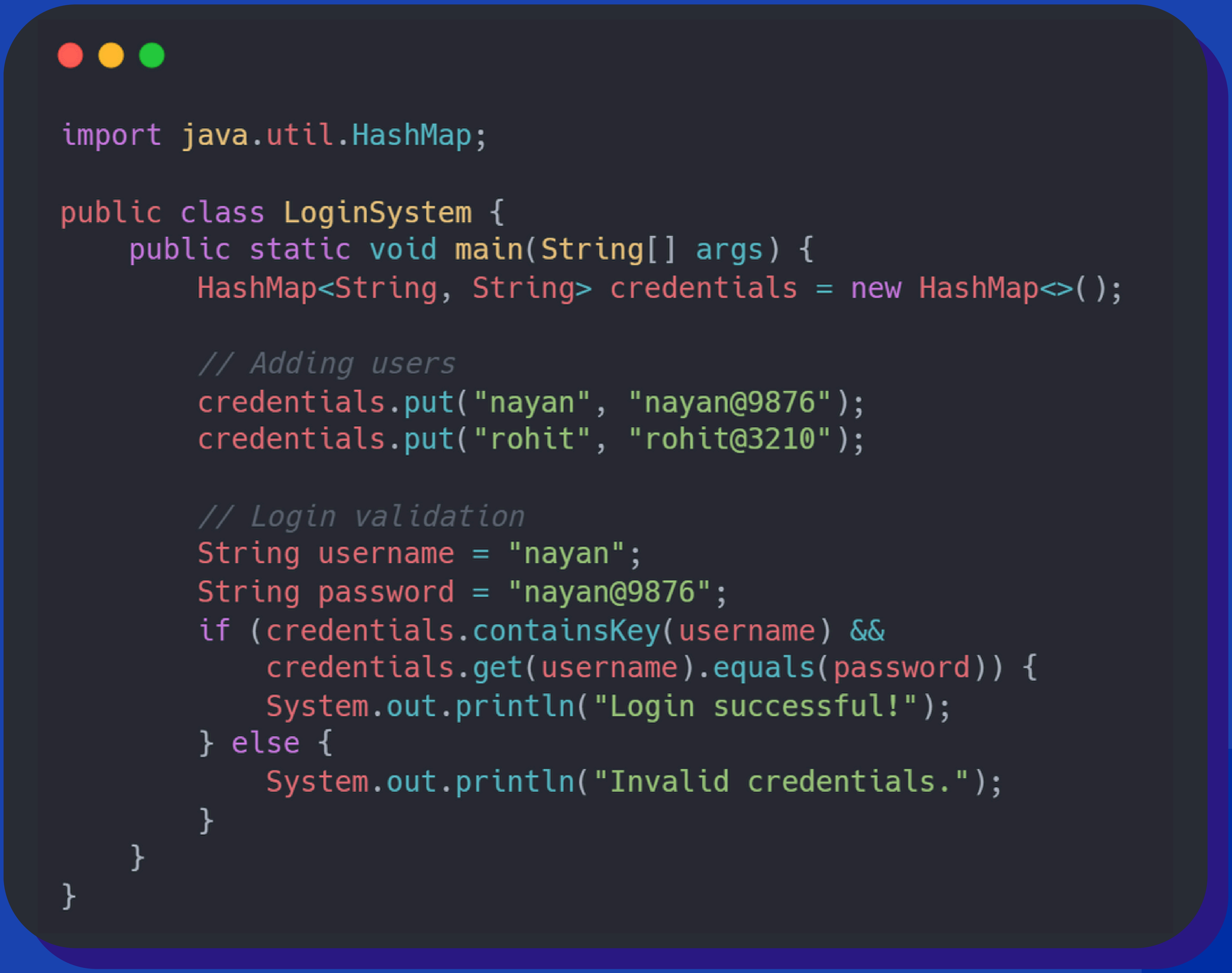
Common Implementations /extends of Map

HashMap (Class)

- **Backed by:** Hash table.
- **Order:** No guaranteed order of keys.
- **Null Support:** Allows one null key and multiple null values.
- **Performance:** Fast insertion and lookup (O(1) in most cases).
- **Use Case:** General-purpose map for non-thread-safe applications.

Let's Swipe Right →

HashMap (Class)



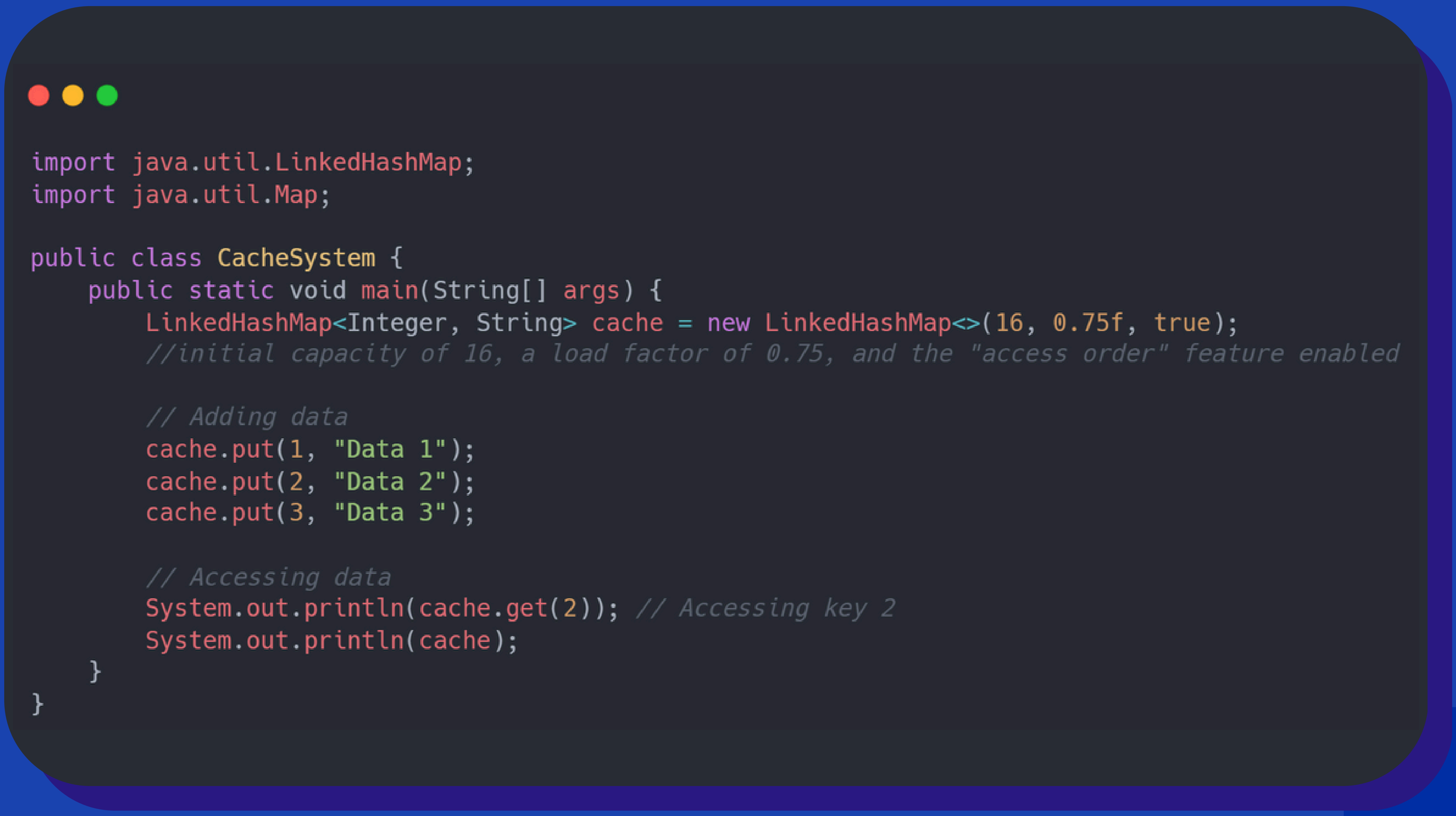
```java
import java.util.HashMap;

public class LoginSystem {
    public static void main(String[] args) {
        HashMap<String, String> credentials = new HashMap<>();

        // Adding users
        credentials.put("nayan", "nayan@9876");
        credentials.put("rohit", "rohit@3210");

        // Login validation
        String username = "nayan";
        String password = "nayan@9876";
        if (credentials.containsKey(username) &&
            credentials.get(username).equals(password)) {
            System.out.println("Login successful!");
        } else {
            System.out.println("Invalid credentials.");
        }
    }
}
```

Common Implementations /extends of Map

LinkedHashMap (Class)

- **Extends:** HashMap with predictable iteration order.
- **Order:** Maintains insertion order or access order (if configured).
- **Performance:** Slightly slower than HashMap due to ordering overhead.
- **Use Case:** When you need predictable iteration order for keys or values.

LinkedHashMap (Class)



```java
import java.util.LinkedHashMap;
import java.util.Map;

public class CacheSystem {
    public static void main(String[] args) {
        LinkedHashMap<Integer, String> cache = new LinkedHashMap<>(16, 0.75f, true);
        //initial capacity of 16, a load factor of 0.75, and the "access order" feature enabled

        // Adding data
        cache.put(1, "Data 1");
        cache.put(2, "Data 2");
        cache.put(3, "Data 3");

        // Accessing data
        System.out.println(cache.get(2)); // Accessing key 2
        System.out.println(cache);
    }
}
```

Let’s Swipe Right

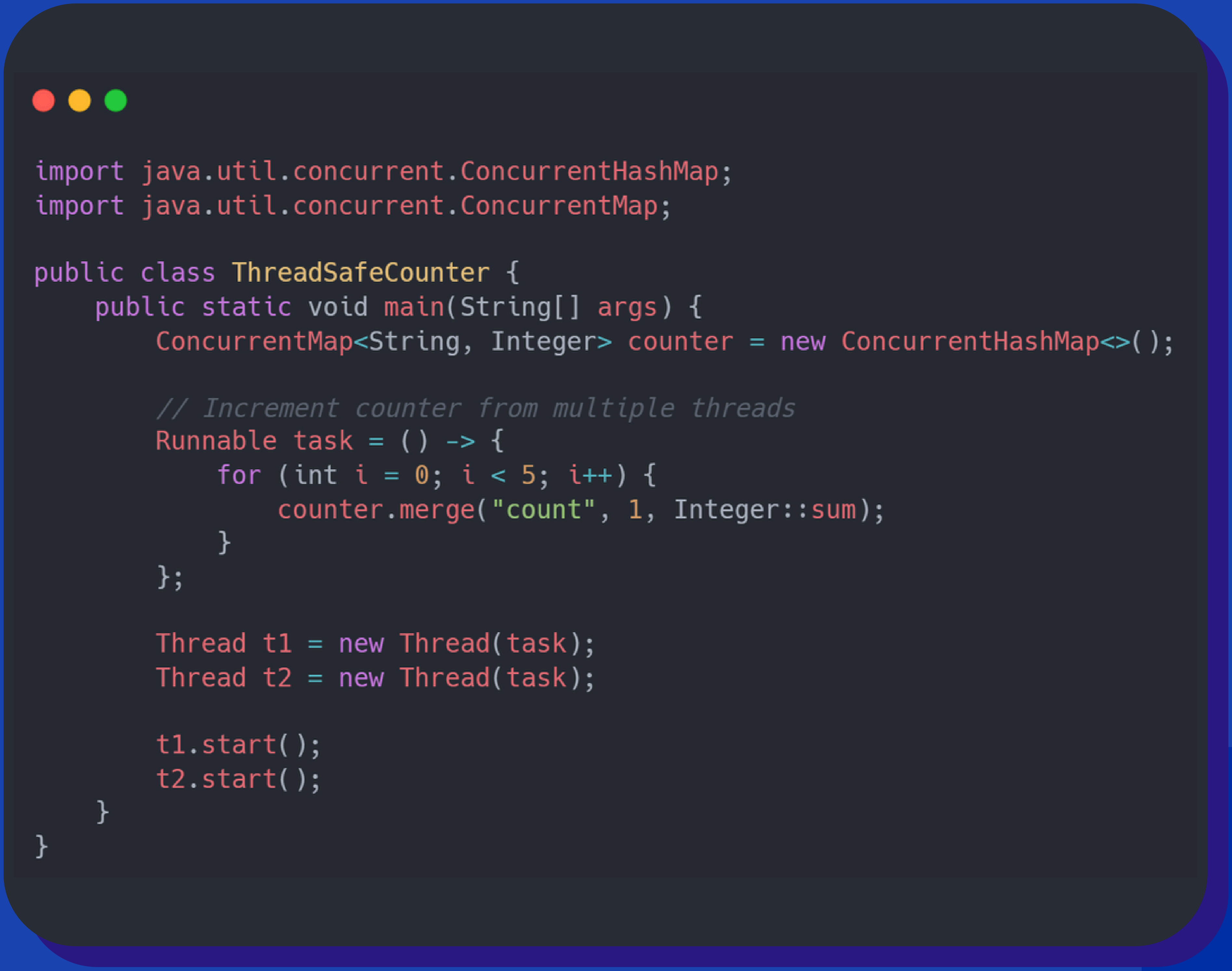
Common Implementations /extends of Map

ConcurrentMap (Interface)

- **Extends:** Map.
- **Purpose:** Provides atomic operations for thread-safe maps.
- **Implementation:** ConcurrentHashMap is the primary implementation.
- **Use Case:** Ideal for multithreaded environments where thread safety is a priority.

Let’s Swipe Right

ConcurrentMap (Interface)



```java
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ConcurrentMap;

public class ThreadSafeCounter {
    public static void main(String[] args) {
        ConcurrentMap<String, Integer> counter = new ConcurrentHashMap<>();

        // Increment counter from multiple threads
        Runnable task = () -> {
            for (int i = 0; i < 5; i++) {
                counter.merge("count", 1, Integer::sum);
            }
        };

        Thread t1 = new Thread(task);
        Thread t2 = new Thread(task);

        t1.start();
        t2.start();
    }
}
```

Nayankumar Dhome nayankumardhome@gmail.com

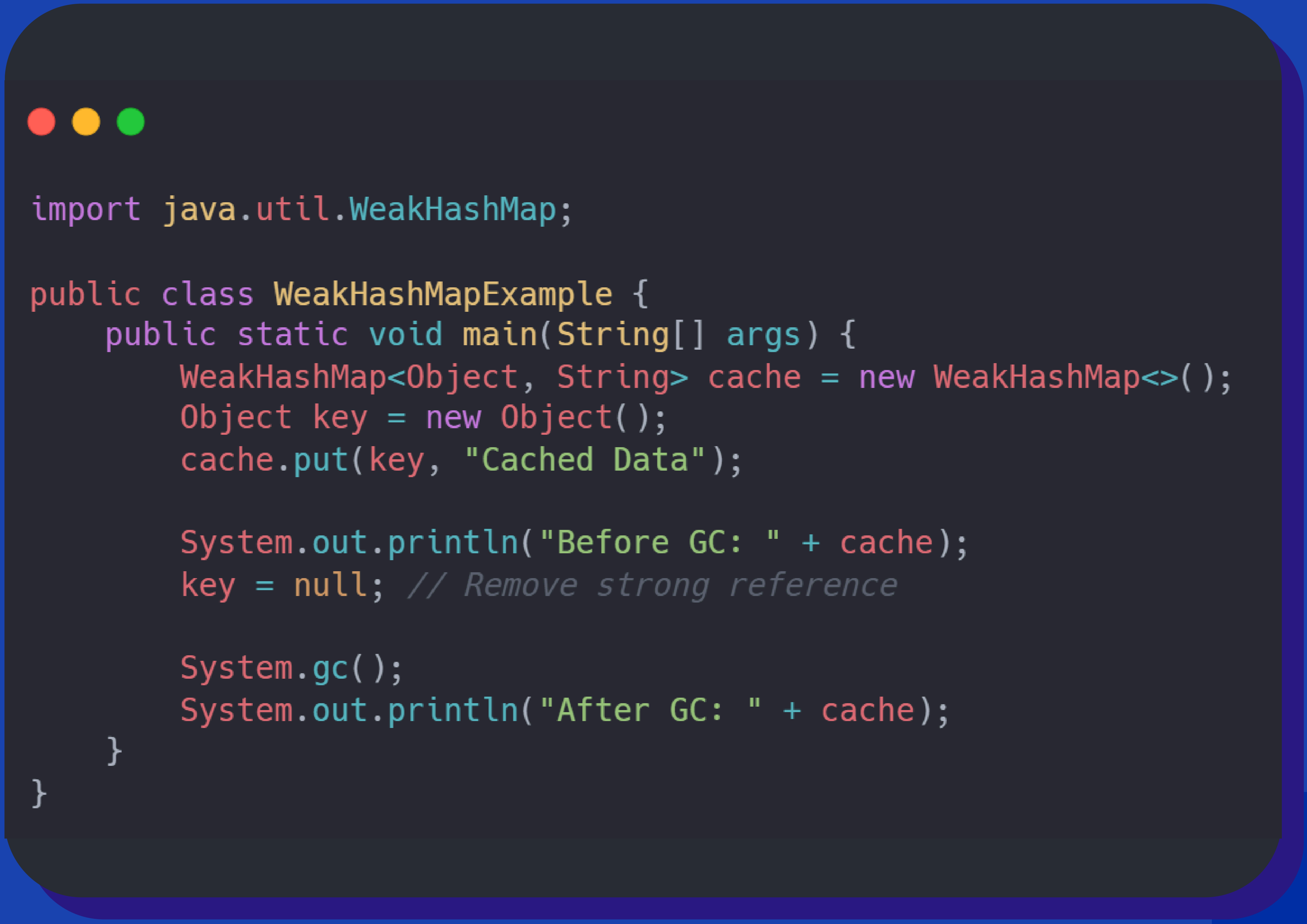
Common Implementations /extends of Map

WeakHashMap (Class)

- **Purpose:** Uses weak references for keys, allowing garbage collection when no strong reference exists to the key.
- **Use Case:** Caches or temporary mappings where memory-sensitive cleanup is needed.

Let's Swipe Right

WeakHashMap (Class)



```java
import java.util.WeakHashMap;

public class WeakHashMapExample {
    public static void main(String[] args) {
        WeakHashMap<Object, String> cache = new WeakHashMap<>();
        Object key = new Object();
        cache.put(key, "Cached Data");

        System.out.println("Before GC: " + cache);
        key = null; // Remove strong reference

        System.gc();
        System.out.println("After GC: " + cache);
    }
}
```

Let’s Swipe Right →

Nayankumar Dhome nayankumardhome@gmail.com

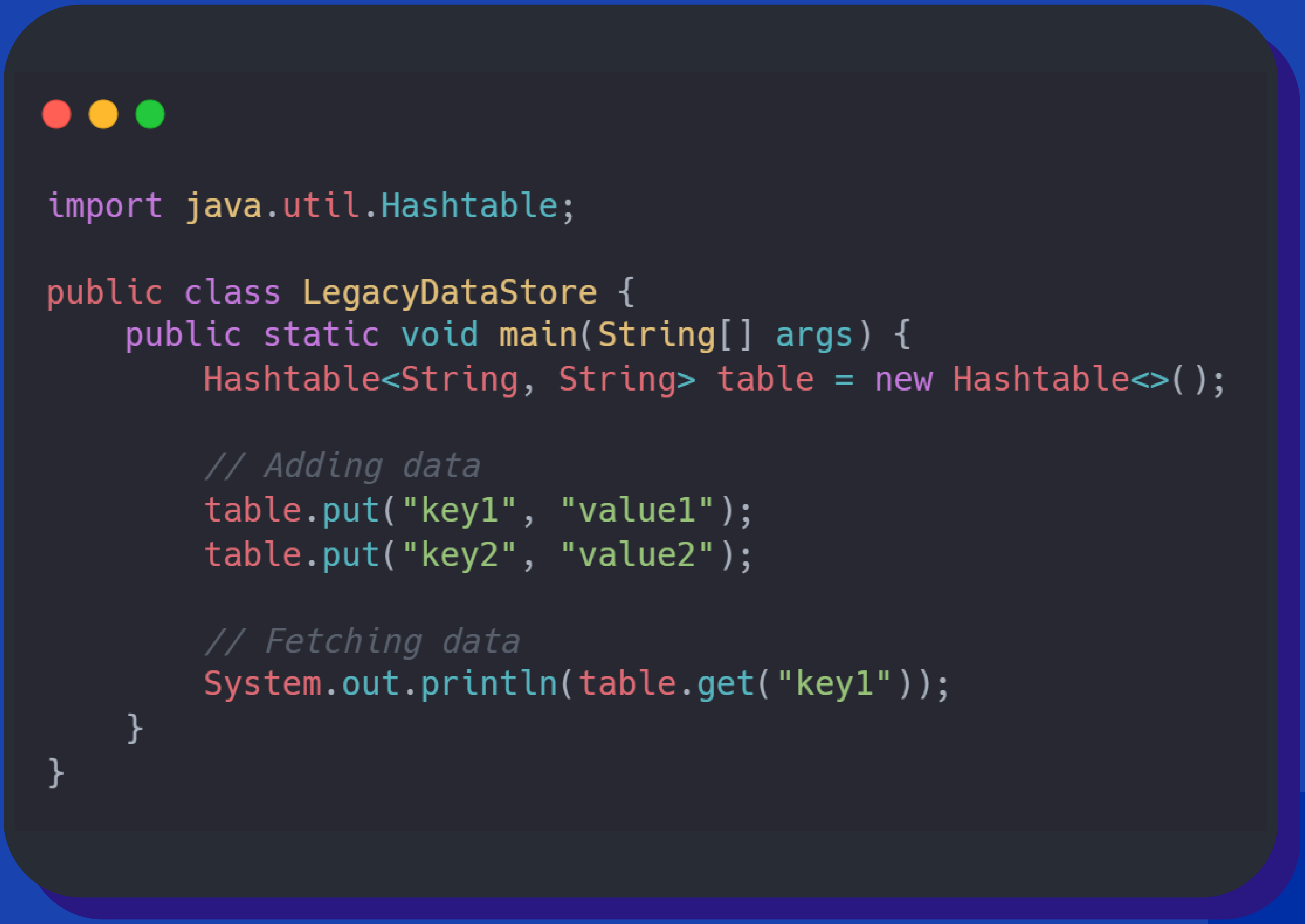
Common Implementations /extends of Map

Hashtable (Class)

- **Purpose:** Legacy synchronized implementation of Map.
- **Order:** No guaranteed order of keys.
- **Null Support:** Does not allow null keys or values.
- **Performance:** Slower than HashMap due to synchronization overhead.
- **Use Case:** Avoid unless thread safety is required in legacy applications.

Let's Swipe Right →

Hashtable (Class)



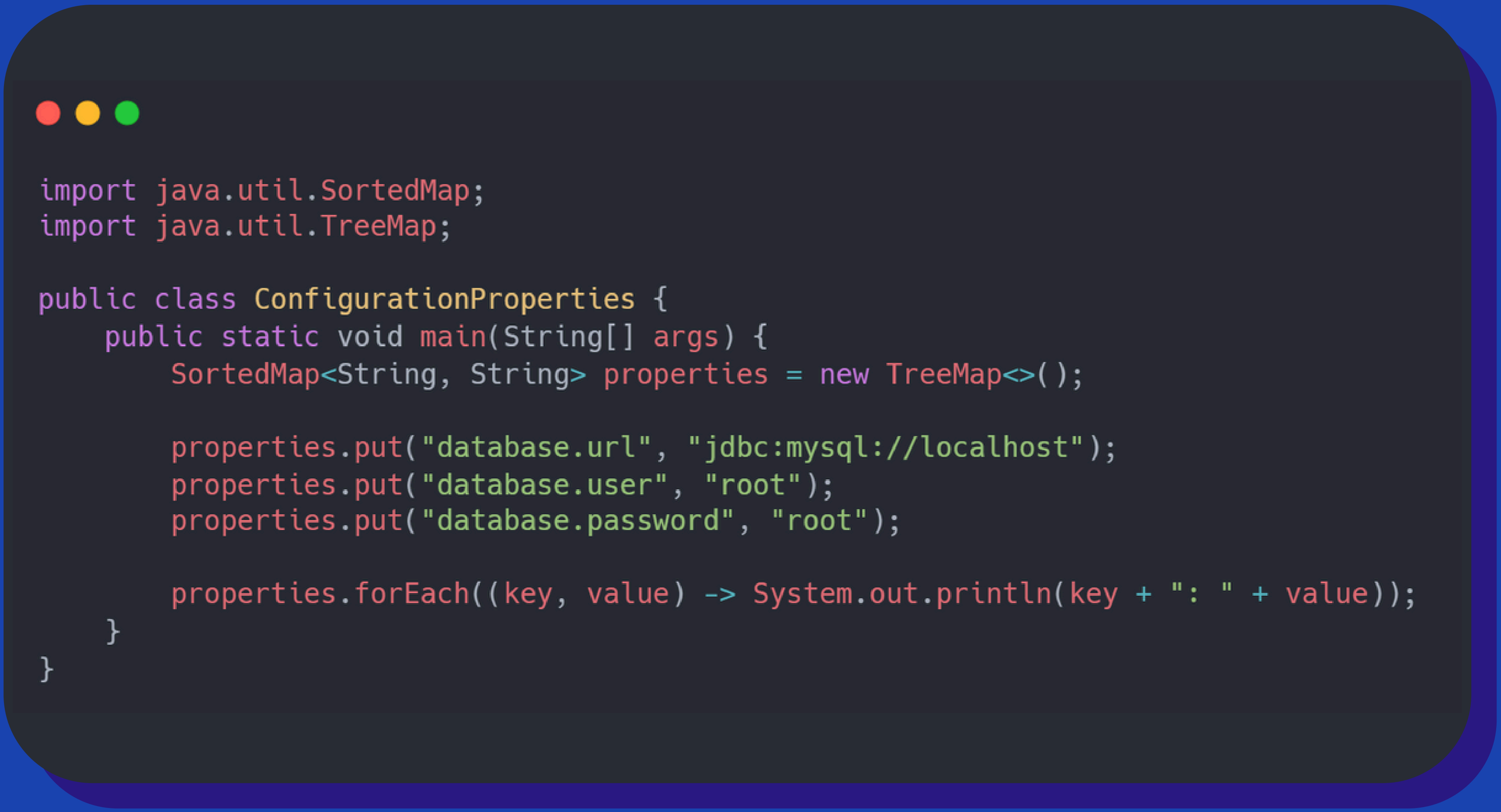
```java
import java.util.Hashtable;

public class LegacyDataStore {
    public static void main(String[] args) {
        Hashtable<String, String> table = new Hashtable<>();

        // Adding data
        table.put("key1", "value1");
        table.put("key2", "value2");

        // Fetching data
        System.out.println(table.get("key1"));
    }
}
```

Common Implementations /extends of Map

SortedMap (Interface)

- **Extends:** Map to provide sorting capabilities.
- **Implementation:**
  - ***TreeMap:*** Automatically sorts keys in natural order or based on a custom comparator.
- **Use Case:** When sorted keys are essential for your application logic.

SortedMap (Interface)



```java
import java.util.SortedMap;
import java.util.TreeMap;

public class ConfigurationProperties {
    public static void main(String[] args) {
        SortedMap<String, String> properties = new TreeMap<>();

        properties.put("database.url", "jdbc:mysql://localhost");
        properties.put("database.user", "root");
        properties.put("database.password", "root");

        properties.forEach((key, value) -> System.out.println(key + ": " + value));
    }
}
```

Let’s Swipe Right →

Nayankumar Dhome nayankumardhome@gmail.com

Key Takeaways

- Use HashMap for general-purpose needs.
- Opt for TreeMap or LinkedHashMap if ordering is essential.
- Leverage ConcurrentHashMap for multithreaded scenarios.
- Specialized maps like WeakHashMap or IdentityHashMap cater to unique use cases.

Let's Swipe Right →

Important Note

For this post, we've only considered classes, interfaces, and abstract classes that directly extend or implement the Map interface. In future posts, we will discuss additional derived classes and specialized implementations in detail.

Let's Swipe Right →

Nayankumar Dhome nayankumardhome@gmail.com

Find this useful?

Like and repost this post with your connections.

Let's Connected