

Github:

<https://github.com/keiakihito/CS5800-HW6>

Driver

```
~/Dropbox/Academic/CalPolyPomona/2025/2025Fall/CS5800/HW/HW6 main*
> mvn exec:java -Dexec.mainClass=chat.Driver

[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.calpoly:cs5800-2025-hw6-chatapp >-----
[INFO] Building cs5800-2025-hw6-chatapp 1.0.0
[INFO]    from pom.xml
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- exec:3.6.1:java (default-cli) @ cs5800-2025-hw6-chatapp ---
== Basic messaging through mediator ==
Taro history:
  Aiko (2025-11-11T07:16:23.546543Z) -> Hi everyone!
  Taro (2025-11-11T07:16:23.546720Z) -> Hey Aiko
Jiro history:
  Aiko (2025-11-11T07:16:23.546543Z) -> Hi everyone!

== Blocking demo ==
Jiro history after blocking Taro:
  Aiko (2025-11-11T07:16:23.546543Z) -> Hi everyone!

== Undo last message (Memento)==
Aiko history before undo:
  Aiko (2025-11-11T07:16:23.546543Z) -> Hi everyone!
  Taro (2025-11-11T07:16:23.546720Z) -> Hey Aiko
  Taro (2025-11-11T07:16:23.548557Z) -> Lunch?
  Aiko (2025-11-11T07:16:23.548687Z) -> Oops, wrong person
Aiko history after undo:
  Aiko (2025-11-11T07:16:23.546543Z) -> Hi everyone!
  Taro (2025-11-11T07:16:23.546720Z) -> Hey Aiko
  Taro (2025-11-11T07:16:23.548557Z) -> Lunch?

== Iterator filtered view ==
Aiko -> [Taro, Jiro] : Hi everyone!
Taro -> [Aiko] : Hey Aiko
Taro -> [Jiro, Aiko] : Lunch?
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time:  0.227 s
[INFO] Finished at: 2025-11-10T23:16:23-08:00
[INFO] -----
```

src > main > java > chat > Driver.java > Driver > main(String[])

```
1 package chat;
2
3 import java.util.Iterator;
4 import java.util.List;
5 import java.util.stream.Collectors;
6
7 /** Simple driver showcasing mediator + memento behavior. */
8 public class Driver {
9     public static void main(String[] args) {
10         ChatServer server = new ChatServer();
11         User aiko = new User(name: "Aiko");
12         User taro = new User(name: "Taro");
13         User jiro = new User(name: "Jiro");
14
15         // Register users with the mediator
16         aiko.join(server);
17         taro.join(server);
18         jiro.join(server);
19
20         System.out.println(x: "== Basic messaging through mediator ==");
21         aiko.sendToMany(List.of(taro, jiro), content: "Hi everyone!");
22         taro.sendTo(aiko, content: "Hey Aiko");
23
24         printHistory(label: "Taro history", taro);
25         printHistory(label: "Jiro history", jiro);
26
27         System.out.println(x: "\n== Blocking demo ==");
28         jiro.block(taro);
29         taro.sendToMany(List.of(jiro, aiko), content: "Lunch?");
30         printHistory(label: "Jiro history after blocking Taro", jiro);
31
32         System.out.println(x: "\n== Undo last message (Memento) ==");
33         aiko.sendTo(jiro, content: "Oops, wrong person");
34         printHistory(label: "Aiko history before undo", aiko);
35         aiko.undoLastMessage();
36         printHistory(label: "Aiko history after undo", aiko);
37
38         System.out.println(x: "\n== Iterator filtered view ==");
39         Iterator<Message> aikoWithTaro = aiko.historyFor(taro);
40         while (aikoWithTaro.hasNext()) {
41             Message message = aikoWithTaro.next();
42             System.out.printf(format: "%s -> %s : %s\n",
43                 message.getSender().getName(),
44                 message.getRecipients().stream().map(User::getName).collect(Collectors.toList()),
45                 message.getContent());
46         }
47
48
49     private static void printHistory(String label, User user) {
50         System.out.println(label + ":");
51         user.getHistory().iterator(user).forEachRemaining(message ->
52             System.out.printf(format: "    %s (%s) -> %s\n",
53                 message.getSender().getName(),
```

```

1  package chat;
2
3  import java.util.ArrayDeque;
4  import java.util.ArrayList;
5  import java.util.Deque;
6  import java.util.Iterator;
7  import java.util.List;
8  import java.util.Objects;
9
10 /** Stores messages for a user. */
11 public class ChatHistory implements IterableByUser {
12     private final List<Message> entries = new ArrayList<>();
13     private final Deque<MessageMemento> sentSnapshots = new ArrayDeque<>();
14
15     public void append(Message message) {
16         if (shouldSkipAppend(message)) {
17             return;
18         }
19         entries.add(message);
20     }
21
22     private boolean shouldSkipAppend(Message message) {
23         if (message == null) {
24             System.out.println(x: "Message is null, skipping append");
25             return true;
26         }
27         return false;
28     }
29
30     public Message lastMessage() {
31         if (hasNoMessages()) {
32             return null;
33         }
34         return getLastEntry();
35     }
36
37     private Message getLastEntry() {
38         return entries.get(entries.size() - 1);
39     }
40
41     private boolean hasNoMessages() {
42         return entries.isEmpty();
43     }
44
45     public void saveSnapshot(MessageMemento memento) {
46         if (memento == null) {
47             System.out.println(x: "Memento is null, skipping snapshot save");
48             return;
49         }
50         sentSnapshots.push(memento);
51     }
52

```

```

53     public MessageMemento lastSentSnapshot() {
54         return sentSnapshots.peek();
55     }
56
57     public MessageMemento removeLastSnapshot() {
58         if (sentSnapshots.isEmpty()) {
59             return null;
60         }
61         return sentSnapshots.pop();
62     }
63
64     public boolean removeMessage(Message target) {
65         if (target == null) {
66             System.out.println(X: "Target message is null, skipping removal");
67             return false;
68         }
69         Iterator<Message> iterator = entries.iterator();
70         while (iterator.hasNext()) {
71             Message entry = iterator.next();
72             if (messagesMatch(entry, target)) {
73                 iterator.remove();
74                 return true;
75             }
76         }
77         return false;
78     }
79
80     private boolean messagesMatch(Message first, Message second) {
81         return Objects.equals(first.getSender(), second.getSender())
82             && Objects.equals(first.getRecipients(), second.getRecipients())
83             && Objects.equals(first.getTimestamp(), second.getTimestamp())
84             && Objects.equals(first.getContent(), second.getContent());
85     }
86
87     @Override
88     public Iterator<Message> iterator(User userToSearchWith) {
89         return new SearchMessagesByUser(entries, userToSearchWith);
90     }
91 }
92

```

```

1  package chat;
2
3  import java.util.Iterator;
4
5  public interface IterableByUser {
6      Iterator<Message> iterator(User userToSearchWith);
7  }
8

```

src > main > java > chat >  Message.java >  Message >  restore(MessageMemento)

```
1  package chat;
2
3  import java.time.Instant;
4  import java.util.List;
5  import java.util.Objects;
6
7  /** Value object for chat messages. */
8  public class Message {
9      private final User sender;
10     private final List<User> recipients;
11     private final Instant timestamp;
12     private final String content;
13
14     public Message(User sender, List<User> recipients, Instant timestamp, String content) {
15         validateNotNull(sender, parameterName: "sender");
16         validateNotNull(recipients, parameterName: "recipients");
17         validateNotNull(timestamp, parameterName: "timestamp");
18         validateNotNull(content, parameterName: "content");
19
20         this.sender = sender;
21         this.recipients = recipients;
22         this.timestamp = timestamp;
23         this.content = content;
24     }
25
26     private void validateNotNull(Object value, String parameterName) {
27         if (value == null) {
28             throw new IllegalArgumentException(parameterName + " cannot be null");
29         }
30     }
31
32     public User getSender() {
33         return sender;
34     }
35
36     public List<User> getRecipients() {
37         return recipients;
38     }
39
40     public Instant getTimestamp() {
41         return timestamp;
42     }
43
44     public String getContent() {
45         return content;
46     }
47
48     public MessageMemento createMemento() {
49         return new MessageMemento(timestamp, content);
50     }
51
52     public Message restore(MessageMemento memento) {
53         validateNotNull(memento, parameterName: "memento");
54         return new Message(sender, recipients, memento.getTimestamp(), memento.getContent());
55     }
56 }
57
```


src > main > java > chat > SearchMessagesByUser.java > SearchMessagesByUser > findNextMatchingMes

```
1 package chat;
2
3 import java.util.Iterator;
4 import java.util.List;
5 import java.util.NoSuchElementException;
6
7 /** Iterator iterating through messages matching a specific user. */
8 public class SearchMessagesByUser implements Iterator<Message> {
9     private final List<Message> history;
10    private final User target;
11    private int cursor = 0;
12    private Message nextMatch;
13
14    public SearchMessagesByUser(List<Message> history, User target) {
15        this.history = history;
16        this.target = target;
17        updateNextMatch();
18    }
19
20    @Override
21    public boolean hasNext() {
22        return nextMatch != null;
23    }
24
25    @Override
26    public Message next() {
27        ensureNextMatchAvailable();
28        Message current = nextMatch;
29        updateNextMatch();
30        return current;
31    }
32
33    private void ensureNextMatchAvailable() {
34        if (nextMatch == null) {
35            throw new NoSuchElementException("No more matching messages");
36        }
37    }
38
39    private void updateNextMatch() {
40        nextMatch = findNextMatchingMessage();
41    }
42
43    private Message findNextMatchingMessage() {
44        while (walker < history.size()) {
45            Message candidate = history.get(walker++);
46            if (matchesTarget(candidate)) {
47                return candidate;
48            }
49        }
50        return null;
51    }
52
53    private boolean matchesTarget(Message candidate) {
54        return candidate.getSender().equals(target)
55            || candidate.getRecipients().contains(target);
56    }
57 }
```

src > main > java > chat > MessageMemento.java > {} chat

```
1 package chat;
2
3 import java.time.Instant;
4
5 /** Snapshot placeholder. */
6 public class MessageMemento {
7     private final Instant timestamp;
8     private final String content;
9
10    public MessageMemento(Instant timestamp, String content) {
11        this.timestamp = timestamp;
12        this.content = content;
13    }
14
15    public Instant getTimestamp() {
16        return timestamp;
17    }
18
19    public String getContent() {
20        return content;
21    }
22
23    public String getSummary() {
24        return "[" + timestamp + "] " + content;
25    }
26 }
27
```

```

1 package chat;
2
3 import java.util.Collections;
4 import java.util.Iterator;
5 import java.util.List;
6 import java.util.Objects;
7
8 /** Chat user placeholder. */
9 public class User {
10     private final String name;
11     private final ChatHistory history;
12     private ChatServer mediator;
13
14     public User(String name) {
15         this.name = Objects.requireNonNull(name);
16         this.history = new ChatHistory();
17     }
18
19     public String getName() {
20         return name;
21     }
22
23     public ChatHistory getHistory() {
24         return history;
25     }
26
27     public void join(ChatServer server) {
28         Objects.requireNonNull(server, message: "server");
29         this.mediator = server;
30         server.register(this);
31     }
32
33     public void sendTo(User recipient, String content) {
34         sendToMany(Collections.singletonList(recipient), content);
35     }
36
37     public void sendToMany(List<User> recipients, String content) {
38         if (recipients == null || recipients.isEmpty()) {
39             throw new IllegalArgumentException(s: "Recipients required");
40         }
41         ensureMediator().deliver(this, List.copyOf(recipients), content);
42     }
43
44     public void receive(Message message) {
45         history.append(message);
46     }
47
48     public void block(User target) {
49         ensureMediator().block(this, target);
50     }
51
52     public void undoLastMessage() {
53         ensureMediator().undoLast(this);
54     }
55
56     public Iterator<Message> historyFor(User target) {
57         return history.iterator(target);
58     }
59
60     private ChatServer ensureMediator() {
61         if (mediator == null) {
62             throw new IllegalStateException(s: "User is not connected to a chat server");
63         }
64         return mediator;
65     }
66 }
67

```

Unit tests

```
[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Running chat.UserTest
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.049 s -- in chat.UserTest
[INFO] Running chat.ChatServerTest
[INFO] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.012 s -- in chat.ChatServerTest
[INFO] Running chat.MessageMementoTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.009 s -- in chat.MessageMementoTest
[INFO] Running chat.MessageTest
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.002 s -- in chat.MessageTest
[INFO] Running chat.SearchMessagesByUserTest
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.004 s -- in chat.SearchMessagesByUserTest
[INFO] Running chat.ChatHistoryTest
[INFO] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.003 s -- in chat.ChatHistoryTest
[INFO] Results:
[INFO] Tests run: 21, Failures: 0, Errors: 0, Skipped: 0
[INFO] --- jacoco:0.8.10:report (report) @ cs5800-2025-hw6-chatapp ---
[INFO] Loading execution data file /Users/keita-katsumi/Dropbox/Academic/CalPolyPomona/2025/2025Fall/CSS800/HW/HW6/target/jacoco.exec
[INFO] Analyzed bundle 'cs5800-2025-hw6-chatapp' with 7 classes
[INFO] BUILD SUCCESS
[INFO] Total time: 2.019 s
[INFO] Finished at: 2025-11-10T23:24:14-08:00
[INFO] -----
```

```

1 package chat;
2
3 import java.time.Instant;
4 import java.util.List;
5 import org.junit.jupiter.api.Test;
6
7 import static org.junit.jupiter.api.Assertions.assertEquals;
8 import static org.junit.jupiter.api.Assertions.assertFalse;
9 import static org.junit.jupiter.api.Assertions.assertTrue;
10
11 class ChatHistoryTest {
12
13     @Test
14     void appendStoresMessagesInOrder() {
15         ChatHistory history = new ChatHistory();
16         User aiko = new User(name: "Aiko Tokumoto");
17         User taro = new User(name: "Taro Yamada");
18         Message firstMsg = new Message(aiko, List.of(taro), Instant.EPOCH, content: "hi");
19         Message secondMsg = new Message(aiko, List.of(taro), Instant.EPOCH.plusSeconds(secondsToAdd: 5), content: "yo");
20
21         history.append(firstMsg);
22         history.append(secondMsg);
23
24         assertEquals("yo", history.lastMessage().getContent());
25     }
26
27     @Test
28     void iteratorFiltersByUser() {
29         ChatHistory history = new ChatHistory();
30         User aiko = new User(name: "Aiko Tokumoto");
31         User taro = new User(name: "Taro Yamada");
32         Message firstMsg = new Message(aiko, List.of(taro), Instant.EPOCH, content: "hello");
33         Message secondMsg = new Message(aiko, List.of(aiko), Instant.EPOCH, content: "self");
34         history.append(firstMsg);
35         history.append(secondMsg);
36
37         var iterator = history.iterator(taro);
38
39         assertTrue(iterator.hasNext());
40         assertEquals("hello", iterator.next().getContent());
41         assertFalse(iterator.hasNext());
42     }
43
44     @Test
45     void snapshotStackTracksLatestMemento() {
46         ChatHistory history = new ChatHistory();
47         MessageMemento first = new MessageMemento(Instant.EPOCH, content: "first");
48         MessageMemento second = new MessageMemento(Instant.EPOCH.plusSeconds(secondsToAdd: 5), content: "second");
49
50         history.saveSnapshot(first);
51         history.saveSnapshot(second);
52
53         assertEquals(second, history.lastSentSnapshot());
54     }
55 }
56

```

```

1  package chat;
2
3  import java.time.Instant;
4  import java.util.List;
5  import org.junit.jupiter.api.Test;
6
7  import static org.junit.jupiter.api.Assertions.assertEquals;
8  import static org.junit.jupiter.api.Assertions.assertFalse;
9  import static org.junit.jupiter.api.Assertions.assertNull;
10 import static org.junit.jupiter.api.Assertions.assertThrows;
11 import static org.junit.jupiter.api.Assertions.assertTrue;
12
13 class ChatServerTest {
14
15     @Test
16     void registerAddsUserToMediator() {
17         ChatServer server = new ChatServer();
18         User user = new User(name: "Aiko Tokumoto");
19
20         server.register(user);
21
22         assertTrue(server.hasUser(username: "Aiko Tokumoto"), "registered user should be discoverable");
23     }
24
25     @Test
26     void unregisterRemovesUserFromMediator() {
27         ChatServer server = new ChatServer();
28         User user = new User(name: "Aiko Tokumoto");
29
30         server.register(user);
31         server.unregister(user);
32
33         assertFalse(server.hasUser(username: "Aiko Tokumoto"), "user should not remain after unregister");
34     }
35
36     @Test
37     void duplicateRegisterThrows() {
38         ChatServer server = new ChatServer();
39         User user = new User(name: "Aiko Tokumoto");
40
41         server.register(user);
42
43         assertThrows(IllegalArgumentException.class, () -> server.register(user));
44     }
45
46     @Test
47     void deliverNotifiesAllRecipients() {
48         ChatServer server = new ChatServer();
49         TestSinkUser sender = new TestSinkUser(name: "Aiko Tokumoto");
50         TestSinkUser receiver1 = new TestSinkUser(name: "Receiver1");
51         TestSinkUser receiver2 = new TestSinkUser(name: "Receiver2");
52         server.register(sender);
53         server.register(receiver1);
54         server.register(receiver2);
55
56         server.deliver(sender, List.of(receiver1, receiver2), content: "Hi team");
57
58         assertEquals("Hi team", receiver1.lastMessageContent);
59         assertEquals(sender, receiver1.lastSender);
60         assertEquals("Hi team", receiver2.lastMessageContent);
61         assertEquals(sender, receiver2.lastSender);
62     }
63
64     @Test
65     void deliverFailsIfRecipientNotRegistered() {
66         ChatServer server = new ChatServer();
67         TestSinkUser sender = new TestSinkUser(name: "sender");
68         TestSinkUser receiver1 = new TestSinkUser(name: "Receiver1"); // not registered
69         server.register(sender);
70
71         assertThrows(IllegalArgumentException.class,
72             () -> server.deliver(sender, List.of(receiver1), "Hi"),
73             "unregistered recipient should cause failure");
74     }

```

src > test > java > chat >  MessageMementoTest.java > {} chat

```
1 package chat;
2
3 import java.time.Instant;
4 import org.junit.jupiter.api.Test;
5
6 import static org.junit.jupiter.api.Assertions.assertTrue;
7
8 class MessageMementoTest {
9
10     @Test
11     void summaryIncludesTimestampAndContent() {
12         Instant timestamp = Instant.parse(text: "2024-01-01T00:00:00Z");
13         MessageMemento memento = new MessageMemento(timestamp, content: "hello");
14
15         String summary = memento.getSummary();
16
17         assertTrue(summary.contains(s: "hello"));
18         assertTrue(summary.contains(s: "2024-01-01"));
19     }
20 }
21
```

src > test > java > chat > MessageTest.java > MessageTest > restoreProducesCopyFromSnapshot()

```
1 package chat;
2
3 import java.time.Instant;
4 import java.util.List;
5 import org.junit.jupiter.api.Test;
6
7 import static org.junit.jupiter.api.Assertions.assertEquals;
8 import static org.junit.jupiter.api.Assertions.assertNotSame;
9
10 class MessageTest {
11
12     @Test
13     void createMementoCapturesContentAndTimestamp() {
14         Instant now = Instant.now();
15         Message message = new Message(new User(name: "Aiko Tokumoto"), List.of(new User(name: "Taro Yamada")));
16
17         MessageMemento memento = message.createMemento();
18
19         assertEquals(now, memento.getTimestamp());
20         assertEquals("hello", memento.getContent());
21     }
22
23     @Test
24     void restoreProducesCopyFromSnapshot() {
25         User sender = new User(name: "Aiko Tokumoto");
26         List<User> recipients = List.of(new User(name: "Taro Yamada"));
27         Message message = new Message(sender, recipients, Instant.EPOCH, content: "hello");
28         MessageMemento snapshot = new MessageMemento(Instant.EPOCH.plusSeconds(secondsToAdd: 5), content: "hello");
29
30         Message restored = message.restore(snapshot);
31
32         assertNotSame(message, restored, "restore should create a new instance");
33         assertEquals("updated", restored.getContent());
34         assertEquals(Instant.EPOCH.plusSeconds(secondsToAdd: 5), restored.getTimestamp());
35         assertEquals(sender, restored.getSender());
36         assertEquals(recipients, restored.getRecipients());
37     }
38 }
39
```

src > test > java > chat > SearchMessagesByUserTest.java > SearchMessagesByUserTest > iteratorHandle

```
1 package chat;
2
3 import java.time.Instant;
4 import java.util.List;
5 import org.junit.jupiter.api.Test;
6
7 import static org.junit.jupiter.api.Assertions.assertEquals;
8 import static org.junit.jupiter.api.Assertions.assertFalse;
9 import static org.junit.jupiter.api.Assertions.assertTrue;
10
11 class SearchMessagesByUserTest {
12
13     @Test
14     void iteratorReturnsOnlyMessagesWithTargetUser() {
15         User aiko = new User(name: "Aiko Tokumoto");
16         User taro = new User(name: "Taro Yamada");
17         User jiro = new User(name: "Jiro Yoshida");
18         Message m1 = new Message(aiko, List.of(taro), Instant.EPOCH, content: "aiko->taro");
19         Message m2 = new Message(taro, List.of(jiro), Instant.EPOCH.plusSeconds(secondsToAdd: 1), con
20         Message m3 = new Message(jiro, List.of(aiko, taro), Instant.EPOCH.plusSeconds(secondsToAdd: 2
21
22         SearchMessagesByUser iterator = new SearchMessagesByUser(List.of(m1, m2, m3), jiro);
23
24         assertTrue(iterator.hasNext());
25         assertEquals("taro->jiro", iterator.next().getContent());
26         assertTrue(iterator.hasNext());
27         assertEquals("group", iterator.next().getContent());
28         assertFalse(iterator.hasNext());
29     }
30
31     @Test
32     void iteratorHandlesNoMatches() {
33         User aiko = new User(name: "Aiko Tokumoto");
34         User taro = new User(name: "Taro Yamada");
35         User jiro = new User(name: "Jiro Yoshida");
36         Message m1 = new Message(aiko, List.of(taro), Instant.EPOCH, content: "aiko->taro");
37
38         SearchMessagesByUser iterator = new SearchMessagesByUser(List.of(m1), jiro);
39
40         assertFalse(iterator.hasNext());
41     }
42 }
43
```

```

1  package chat;
2
3  import java.util.List;
4  import org.junit.jupiter.api.Test;
5
6  import static org.junit.jupiter.api.Assertions.assertEquals;
7  import static org.junit.jupiter.api.Assertions.assertThrows;
8  import static org.junit.jupiter.api.Assertions.assertTrue;
9
10 class UserTest {
11
12     @Test
13     void joinRegistersUserWithServer() {
14         ChatServer server = new ChatServer();
15         User user = new User(name: "Aiko Tokumoto");
16
17         user.join(server);
18
19         assertTrue(server.hasUser(username: "Aiko Tokumoto"));
20     }
21
22     @Test
23     void sendWithoutServerThrowsIllegalState() {
24         User sender = new User(name: "Aiko Tokumoto");
25         User recipient = new User(name: "Taro Yamada");
26
27         assertThrows(IllegalStateException.class, () -> sender.sendTo(recipient, "hi"));
28     }
29
30     @Test
31     void undoWithoutServerThrowsIllegalState() {
32         User user = new User(name: "Aiko Tokumoto");
33
34         assertThrows(IllegalStateException.class, user::undoLastMessage);
35     }
36
37     @Test
38     void sendRoutesThroughMediator() {
39         ChatServer server = new ChatServer();
40         User sender = new User(name: "Aiko Tokumoto");
41         User recipient = new User(name: "Taro Yamada");
42         sender.join(server);
43         recipient.join(server);
44
45         sender.sendTo(recipient, content: "hello");
46
47         assertEquals("hello", recipient.getHistory().lastMessage().getContent());
48         assertEquals("hello", sender.getHistory().lastMessage().getContent());
49     }
50 }

```