Peer Assisted Study Session

FIT2099 - Week 12

Monash University

Objectives

- Given a code base understanding and drawing a class diagram
- Using OOD techniques to refactor a code base

Estimated Time

Question 1 (10 Minutes)
Question 2 - Question 3 (5 Minutes)
Question 4 (5 Minutes)
Question 5 (15 Minutes)
Question 6 - Question 7 (15 Minutes)

Questions

```
import java.util.*;

class Main {
 public static void main(String[] args) {
   ComposeEmail ce = new ComposeEmail();
   ce.run();
 }
}

class ComposeEmail {
 PlainEditor pe = new PlainEditor();
 GmailTransmitter gt = new GmailTransmitter();
  public void run() {
   while (true) {
     String src = pe.promptLine("From: ");
     String dst = pe.promptLine("To: ");
     String msg = pe.promptLine("Message (single line): ");

     if (src == null || dst == null || msg == null) {
       System.out.println("Aborting...");
```

```
        break;
      }

      gt.transmit(src, dst, msg);
      System.out.println();
    }
  }
}

class PlainEditor {
 Scanner s = new Scanner(System.in);
  /**
  * Obtains the next line from stdin.
  * Returns null if there is nothing in the stream.
  *
  * @param prompt the prompt string
  **/
 public String promptLine(String prompt) {
   System.out.print(prompt);

   if (s.hasNext())
     return s.nextLine();
   else
     return null;
 }
}

class GmailTransmitter {
 public void transmit(String src, String dst, String msg) {
   System.out.printf(
     "[GMAIL] (%s) -> (%s): %s\n"
     , src, dst, msg
     );
 }
}

class MoodleTransmitter {
```

```
 public void transmit(String src, String dst, String msg) {
   System.out.printf(
     "[MOODLE] (%s) -> (%s): %s\n"
     , src, dst, msg
     );
 }
}
```

1. Sketch a UML class diagram for the system

2. Change the program to work with the MoodleTransmitter instead of GmailTransmitter

3. In the same vein as the predefined transmitters, define your own transmitter to simulate sending messages over SMTP. Modify the program to work with your transmitter

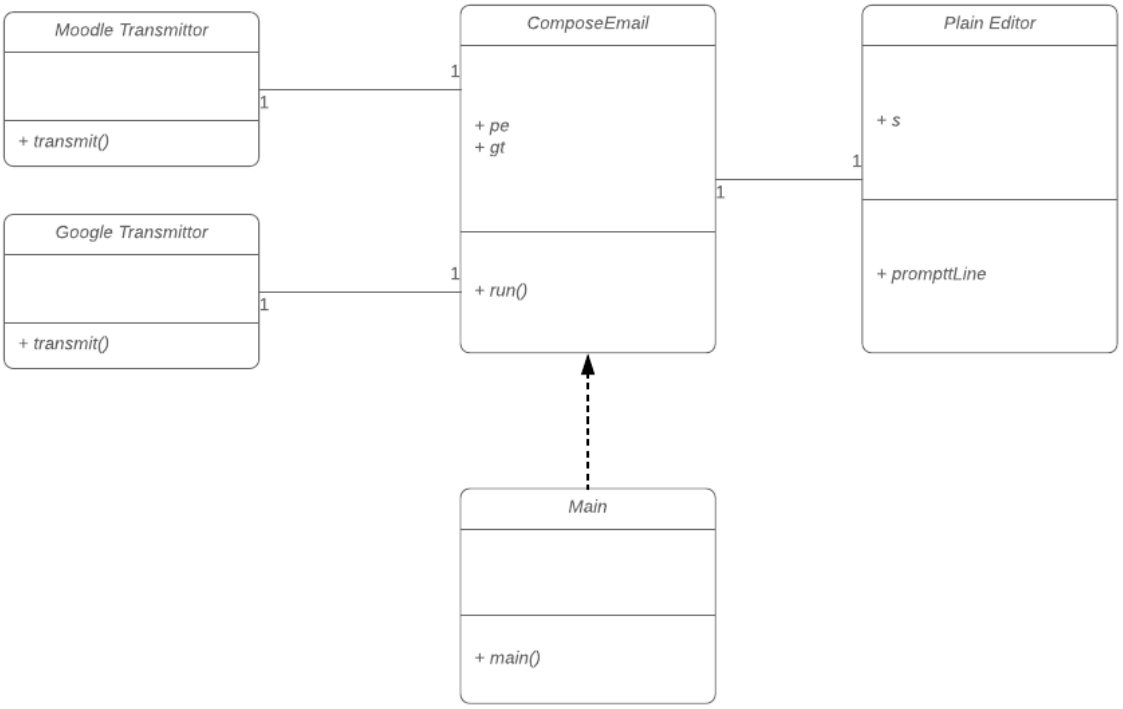
4. What is the problem with this approach to extending the program to work with different transport methods?

5. Propose a better design and sketch its UML class diagram.

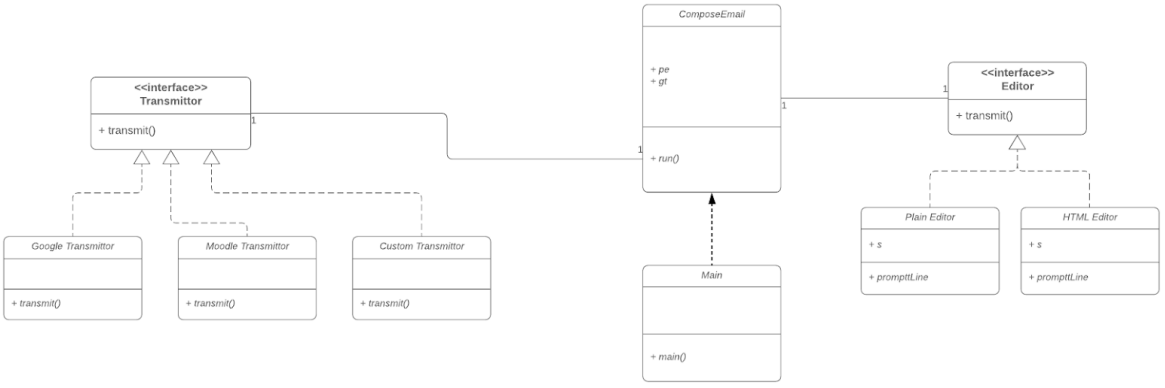
```
import java.util.*;

class Main {
 public static void main(String[] args) {
   ComposeEmail ce = new ComposeEmail();
   ce.run();
 }
}

class ComposeEmail {
 PlainEditor pe = new PlainEditor();
 Transmittor gt = new GmailTransmitter();
  public void run() {
```

```
    while (true) {
      String src = pe.promptLine("From: ");
      String dst = pe.promptLine("To: ");
      String msg = pe.promptLine("Message (single line): ");

      if (src == null || dst == null || msg == null) {
        System.out.println("Aborting...");
        break;
      }

      gt.transmit(src, dst, msg);
      System.out.println();
    }
  }
}

class PlainEditor {
  Scanner s = new Scanner(System.in);
  /**
  * Obtains the next line from stdin.
  * Returns null if there is nothing in the stream.
  *
  * @param prompt the prompt string
  **/
  public String promptLine(String prompt) {
    System.out.print(prompt);

    if (s.hasNext())
      return s.nextLine();
    else
      return null;
  }
}

class GmailTransmitter implements Transmittor{
  public void transmit(String src, String dst, String msg) {
    System.out.printf(
```

```
      "[GMAIL] (%s) -> (%s): %s\n"
      , src, dst, msg
      );
 }
}

class MoodleTransmitter implements Transmittor{
 public void transmit(String src, String dst, String msg) {
   System.out.printf(
     "[MOODLE] (%s) -> (%s): %s\n"
     , src, dst, msg
     );
 }
}

class CustomTransmitter implements Transmittor{
 public void transmit(String src, String dst, String msg) {
   System.out.printf(
     "[CUSTOM] (%s) -> (%s): %s\n"
     , src, dst, msg
     );
 }
}

interface Transmittor{
 public void transmit(String src, String dst, String msg);
}
```



6. Is the editor suffering from the same drawback observed in (3)? If so, propose a design fix and add it to the diagram from (3).



7. Implement the final design.

```
import java.util.*;

class Main {
```

```
 public static void main(String[] args) {
   ComposeEmail ce = new ComposeEmail(new PlainEditor(),new
MoodleTransmitter());
   ce.run();
 }
}

class ComposeEmail {
 private Editor pe;
 private Transmitter gt;
  public ComposeEmail(Editor ed, Transmitter tr){
   pe = ed;
   gt = tr;
 }

 public void run() {
   while (true) {
     String src = pe.promptLine("From: ");
     String dst = pe.promptLine("To: ");
     String msg = pe.promptLine("Message (single line): ");

     if (src == null || dst == null || msg == null) {
       System.out.println("Aborting...");
       break;
     }

     gt.transmit(src, dst, msg);
     System.out.println();
   }
 }
}

class PlainEditor implements Editor {
 Scanner s = new Scanner(System.in);
  /**
  * Obtains the next line from stdin.
  * Returns null if there is nothing in the stream.
```

```
  *
  * @param prompt the prompt string
  **/
 public String promptLine(String prompt) {
   System.out.print(prompt);

   if (s.hasNext())
     return s.nextLine();
   else
     return null;
 }
}

class GmailTransmitter implements Transmitter{
 public void transmit(String src, String dst, String msg) {
   System.out.printf(
     "[GMAIL] (%s) -> (%s): %s\n"
     , src, dst, msg
     );
 }
}

class MoodleTransmitter implements Transmitter{
 public void transmit(String src, String dst, String msg) {
   System.out.printf(
     "[MOODLE] (%s) -> (%s): %s\n"
     , src, dst, msg
     );
 }
}

class CustomTransmitter implements Transmitter{
 public void transmit(String src, String dst, String msg) {
   System.out.printf(
     "[CUSTOM] (%s) -> (%s): %s\n"
     , src, dst, msg
     );
```

```
 }
}

interface Transmitter{
 public void transmit(String src, String dst, String msg);
}

interface Editor{
 public String promptLine(String prompt);
}
```