Gevorderd Programmeren 2019 - 2020 (2e zit)

Examen Practicum

J. Oramas, G. Daneels

1 september 2020

De opgave

In deze opgave ga je een applicatie nabootsen die e-mails kan verzenden en ontvangen. Er worden filters toegepast op de verzonden en ontvangen e-mails om ongeldige e-mails te onderscheppen.

Er zijn verschillende types e-mail, maar elke e-mail bevat een `sender`, `receiver`, `subject` en `content` veld. Specifiek voor de verschillende types e-mail:

- Plain e-mail: dit type e-mailbericht bevat alleen de hiervoor vernoemde velden
- HTML e-mail: dit type e-mailbericht bevat een extra veld `HTML content` met daarin de HTML content
- Attachment e-mail: dit type e-mailbericht bevat een extra veld `attachment` met daarin naam van de (fictieve) attachment

Zoals gezegd kan de e-mailapplicatie e-mails ontvangen en verzenden. Deze e-mails worden echter eerst gefilterd. Filters kunnen worden toegepast bij het verzenden en/of het ontvangen van een email. Dit zijn de mogelijke filters:

- FilterNoContent: deze filter laat geen e-mails met lege content door
- FilterDiffAddresses: deze filter laat geen e-mails door waar het `sender` adres gelijk is aan het `receiver` adres
- FilterNoSpam: deze filter laat geen spam e-mails door, te herkennen aan het adres *earnmoney@easymoney.ng* in het `sender` veld

Als een filter een e-mail afwijst (bij het verzenden of bij het ontvangen), wordt deze naar de prullenbak verwezen. Als een *ontvangen* e-mail niet wordt afgewezen door één van de filters, wordt deze in de inbox geplaatst. Er wordt geen outbox ondersteund in deze e-mailapplicatie.

Implementatie

Voor een uitbreidbare en flexibele implementatie van deze simulatie zijn er de volgende vereisten:

- Er zijn verschillende types e-mail zoals vermeld in de Opgave sectie, afgeleid van de **abstracte** klasse `Mail`. Deze verschillende types bevatten dezelfde basisvelden, maar zowel het HTML type als het Attachment type hebben een extra veld. Een `Mail` moet kunnen worden uitgeprint met de `std::cout` methode (zie main.cpp voorbeeld), maar voor de verschillende types gebeurt dit op een verschillende manier (zie output voorbeeld). Los dit op een polymorfe manier op. Deze methode print het `sender`, `receiver`, `subject` en `content` veld uit, plus een vermelding van het specifieke type en het (eventuele) extra veld.

- De emailapplicatie wordt voorgesteld door de `MailApplication` class. De `MailApplication` heeft volgende functionaliteit:
  - een `addSendFilter` methode voegt filters toe die wordt toegepast bij het verzenden van e-mails
  - een `addReceiveFilter` methode voegt een filter toe die wordt toegepast bij het ontvangen van e-mails
  - de `>` operator laat toe e-mails te verzenden
  - de `<` operator laat toe e-mails te ontvangen
  - de `>>` operator print alle emails in de inbox en trash uit

- Filters worden opgeslagen in 2 containers die objecten van het type `std::function<bool (const std::unique_ptr<Mail>&)>` bevatten. Er zijn filters die worden toegepast bij het verzenden van e-mails en die worden toegepast bij het ontvangen.

- Definieer een type alias, genaamd **t_mails**, voor een `std::vector` van `std::unique_ptr`s naar het `Mail` type. `MailApplication` bevat de datamembers `inbox` en `trash` van het type **t_mails**. In `inbox` worden alleen e-mails opgeslagen die correct ontvangen zijn, in `trash` worden alle e-mails opgeslagen die onderschept zijn door een filter bij het verzenden of ontvangen.

- De `FilterNoContent` en `FilterDiffAddresses` filters moeten elk een **lambda** functie zijn, de `FilterNoSpam` filter moet een **functor** zijn.

Voorbeeld

De **main**-functie moet alvast volgende code bevatten, maar **moet** ook **extra code** bevatten om de implementatie te laten werken:

```
int main(int argc, char *argv[]) {
    try {
        /*
         * Missing code here...
         */

        std::unique_ptr<Mail> mail1 =
        std::make_unique<AttachmentMail>("from@mail1.be", "to@mail1.be",
            "subject1", "Hello World!", "cat.jpg");
        std::cout << mail1 << std::endl;
        std::unique_ptr<Mail> mail2 =
        std::make_unique<HTMLMail>("earnmoney@easymoney.ng", "to@mail2.be",
            "subject2", "Hello World!", "<html>content</html>");
        std::cout << mail2 << std::endl;
        std::unique_ptr<Mail> mail3 =
        std::make_unique<PlainMail>("from@mail3.be", "to@mail3.be",
            "subject3", "Hello World!");
        std::cout << mail3 << std::endl;
        std::unique_ptr<Mail> mail4 =
        std::make_unique<PlainMail>("from@mail4.be", "from@mail4.be",
            "subject4", "");
        std::cout << mail4 << std::endl;

        MailApplication app;
        app.addSendFilter(filterDiffAddresses);
        app.addSendFilter(filterNoEmptyContent);
        app.addReceiveFilter(filterNoSpam);
        app < std::move(mail1) < std::move(mail2)
            < std::move(mail3) > std::move(mail4);
        std::cout << app;
    } catch (exception &e) {
        cout << "Exception: " << e.what() << endl;
    }
    return 0;
}
```

Je vindt deze code ook in het `main.cpp` bestand op Blackboard.

Output

De output van het programma in het main-voorbeeld moet er zo uitzien:

```
Attachment Mail, from from@mail1.be, content: Hello World!, attachment: cat.jpg
HTML Mail, from earnmoney@easymoney.ng, content: Hello World!, HTML: <html>content</html>
Plain Mail, from from@mail3.be, content: Hello World!
Plain Mail, from from@mail4.be, content:
[Receiving] Received a mail from from@mail1.be.
[Receiving] Received a mail from from@mail3.be.
[Inbox] Attachment Mail, from from@mail1.be, content: Hello World!, attachment: cat.jpg
[Inbox] Plain Mail, from from@mail3.be, content: Hello World!
[Trash] HTML Mail, from earnmoney@easymoney.ng, content: Hello World!, HTML: <html>content</html>
[Trash] Plain Mail, from from@mail4.be, content:
```

Opmerkingen

Let op de volgende (zeer belangrijke) zaken:

- Implementeer "minimale" code om de structuur van de besproken entiteiten op te bouwen. Gebruik je tijd dus nuttig om eerst de gevraagde features te implementeren. Zaken die je niet hoeft te schrijven (bv. overbodige constructoren, door de compiler gesynthetiseerde methodes, andere niet gebruikte methodes, ...) laat je best achterwege.
- Gebruik exception handling voor fout-afhandeling. Gebruik `std::invalid_argument` uit `#include <stdexcept>` voor het afhandelen van volgende fouten:
  - Het argument van de `addSendFilter` methode is een `nullptr`.
  - Het argument van de `addReceiveFilter` methode is een `nullptr`.

  Maak je eigen `MailException` exception door af te leiden van `std::logic_error` uit `#include <stdexcept>`. Het moet mogelijk zijn een bericht mee te geven wanneer je de exception gooit.
  De volgende fouten moet je afhandelen met `MailException`:
  - Er wordt een e-mail verzonden, maar er zijn nog geen filters toegevoegd die kunnen worden toegepast op e-mails die worden verzonden.
  - Er wordt een e-mail ontvangen, maar er zijn nog geen filters toegevoegd die kunnen worden toegepast op e-mails die worden ontvangen.
- Gebruik de standard library waar nodig, nuttig en/of warm aanbevolen.
- Vergeet niet `const`-correct te zijn!
- Voorzie je code van duidelijke maar summiere commentaar en schrijf in **elk bestand je naam en rolnummer**.

Veel succes!