EJERCICIO 1

En sí Java no proporciona un mecanismo de exportación selectiva similar al que existe en Eiffel, ya que el control de acceso se basa en modificadores de acceso, como ‘public’, ‘protected’ o ‘private’. Estos modificadores operan a nivel de clase, no a nivel de método específico.

El patrón que hemos llevado a cabo se basa en el patrón método fábrica, en el que los tres tipos de clientes A, B y C, heredan de una interfaz Cliente que define todo tipo de cliente. Por otro lado, están las clases FabricaX, FabricaXA, FabricaXB y FabricaXC, en la que FabricaX es la clase base abstracta que contiene los métodos en común que todas las clases van a poder acceder, a partir de esa, surgen las otras fábricas que extienden a esa inicial, aportando en cada caso los métodos específicos para cada cliente. Por lo que la clase ClienteA va a poder acceder a todos los métodos públicos de FabricaX y a todos los métodos de FabricaXA, y así con cada tipo de cliente.

La ventaja de la solución propuesta es que te aseguras de que todo cliente puede acceder únicamente a los métodos que tiene permiso, ya que la creación de una clase por fábrica y por cliente limita ese aspecto con total seguridad. Sin embargo, tiene un inconveniente, y es que se crean demasiadas clases con poca información a nivel de código, meramente organizativas.

EJERCICIO 3

El patrón de diseño aplicable a este ejercicio es el patrón de estrategia, gracias al cual podemos crear distintas clases SortXXX dependiendo del atributo del mensaje que queremos tener en cuenta para realizar la ordenación. Para no repetir código, el esquema utiliza una función sort() en Mailbox que implementa el algoritmo, sólo refiriéndose a la estrategia el método before(). Gracias a este patrón podemos crear las estrategias que queramos, incluso en casos donde Email se extienda con más atributos. Simplemente haría falta crear un nuevo criterio de comparación en una nueva clase SortXXX.

A screenshot of a computer

Description automatically generated

Clases:

Mailbox:

import java.util.ArrayList;

import static java.util.Collections.swap;

public class Mailbox {

private ArrayList<Email> email = new ArrayList<>();

private SortStrategy sortStrategy;

void setSortStrategy(SortStrategy sortStrategy) {

     this.sortStrategy = sortStrategy;

}

public void show(){

     sort();

     for (Email email : email) {

         System.out.println(email);

     }

}

private void sort() {

     for (int i = 2; i < email.size(); i++) {

         for (int j = email.size()-1; j >= i; j-- ){

             if (before(email.get(j-1),email.get(j)) ){

                 swap(email,j,j-1);

             }

         }

     }

}

private boolean before(Email email, Email email1) {

     return sortStrategy.sort(email, email1);

}

public void addEmail(Email email) {

     this.email.add(email);

}

}

Email:

import java.text.SimpleDateFormat;

import java.util.Date;

public class Email {

private String from;

private String subject;

private String date;

private boolean priority;

private String text;

public Email(String from, String subject, String date, boolean priority, String text) {

     this.from = from;

     this.subject = subject;

     this.date = date;

     this.priority = priority;

     this.text = text;

}

public long getDate() {

     String dateString = this.date;

     SimpleDateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy");

     try{

         Date date = dateFormat.parse(dateString);

         return date.getTime();

     } catch (Exception e) {

         return 0;

     }

}

public String getFrom() {return from;}

public String getSubject() {

     return subject;

}

public boolean getPriority() {

     return priority;

}

public String getText() {

     return text;

}

@Override

public String toString() {

     return "From: " + from + "\n" +

             "Subject: " + subject + "\n" +

             "Date: " + date + "\n" +

             "Priority: " + priority + "\n" +

             "Text: " + text + "\n";

}

}

Interfaz SortStrategy:

public interface SortStrategy {

public boolean sort(Email email1, Email email2);

}

SortDate:

public class SortDate implements SortStrategy  {

@Override

public boolean sort(Email email1, Email email2) {

     return email1.getDate()-email2.getDate() > 0;

}

}

SortFrom:

public class SortFrom implements SortStrategy{

@Override

public boolean sort(Email email1, Email email2) {

     String from1 = email1.getFrom();

     String from2 = email2.getFrom();

     int comparisonResult = from1.compareTo(from2); //should be earlier (a.b) < 0, should be after (b.a) > 0 (so should be swapped)

     return comparisonResult > 0;

}

}

SortPriority:

public class SortPriority implements SortStrategy {

@Override

public boolean sort(Email email1, Email email2) {

     return email1.getPriority();

}

}

SortSubject:

public class SortSubject implements  SortStrategy{

@Override

public boolean sort(Email email1, Email email2) {

     String subject1 = email1.getSubject();

     String subject2 = email2.getSubject();

     int comparisonResult = subject1.compareTo(subject2); //should be earlier (a.b) < 0, should be after (b.a) > 0 (so should be swapped)

     return comparisonResult > 0;

}

}