4.3 Логирование

Это задачка совмещает тренировку по материалу предыдущих двух модулей – необходимо разобраться и написать объект-ориентированный код и при этом коснуться свежих тем – исключений и логирования.

Дан набор классов, описывающих работу гипотетической почтовой системы.

Для начала рассмотрим код, описывающий все используемые сущности.

```
/*
Интерфейс: сущность, которую можно отправить по почте.
У такой сущности можно получить от кого и кому направляется письмо.
*/
public static interface Sendable {
    String getFrom();
    String getTo();
}
```

У Sendable есть два наследника, объединенные следующим абстрактным классом:

```
/*
Абстрактный класс,который позволяет абстрагировать логику хранения
источника и получателя письма в соответствующих полях класса.
*/
public static abstract class AbstractSendable implements Sendable {

    protected final String from;
    protected final String to;

    public AbstractSendable(String from, String to) {
        this.from = from;
        this.to = to;
    }

    @Override
    public String getFrom() {
        return from;
    }

    @Override
    public String getTo() {
        return to;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        AbstractSendable that = (AbstractSendable) o;

        if (!from.equals(that.from)) return false;
        if (!to.equals(that.to)) return false;

        return true;
    }

}
```

Первый класс описывает обычное письмо, в котором находится только текстовое сообщение.

```
/*
Письмо, у которого есть текст, который можно получить с помощью метода `getMessage`
*/
public static class MailMessage extends AbstractSendable {

    private final String message;

    public MailMessage(String from, String to, String message) {
        super(from, to);
        this.message = message;
    }

    public String getMessage() {
        return message;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        if (!super.equals(o)) return false;

        MailMessage that = (MailMessage) o;

        if (message != null ? !message.equals(that.message) : that.message != null) return false;

        return true;
    }

}
```

Второй класс описывает почтовую посылку:

```
/*
Посылка, содержимое которой можно получить с помощью метода `getContent`
*/
public static class MailPackage extends AbstractSendable {
    private final Package content;

    public MailPackage(String from, String to, Package content) {
        super(from, to);
        this.content = content;
    }

    public Package getContent() {
        return content;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        if (!super.equals(o)) return false;

        MailPackage that = (MailPackage) o;

        if (!content.equals(that.content)) return false;

        return true;
    }

}
```

При этом сама посылка описывается следующим классом:

```
/*
Класс, который задает посылку. У посылки есть текстовое описание содержимого и целочисленная ценность.
*/
public static class Package {
    private final String content;
    private final int price;

    public Package(String content, int price) {
        this.content = content;
        this.price = price;
    }

    public String getContent() {
        return content;
    }

    public int getPrice() {
        return price;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        Package aPackage = (Package) o;

        if (price != aPackage.price) return false;
        if (!content.equals(aPackage.content)) return false;

        return true;
    }
}
```

Теперь рассмотрим классы, которые моделируют работу почтового сервиса:

```
/*
Интерфейс, который задает класс, который может каким-либо образом обработать почтовый объект.
*/
public static interface MailService {
    Sendable processMail(Sendable mail);
}

/*
Класс, в котором скрыта логика настоящей почты
*/
public static class RealMailService implements MailService {

    @Override
    public Sendable processMail(Sendable mail) {
        // Здесь описан код настоящей системы отправки почты.
        return mail;
    }
}
```

Вам необходимо описать набор классов, каждый из которых является MailService:

1) UntrustworthyMailWorker – класс, моделирующий ненадежного работника почты, который вместо того, чтобы передать почтовый объект непосредственно в сервис почты, последовательно передает этот объект набору третьих лиц, а затем, в конце концов, передает получившийся объект непосредственно экземпляру RealMailService. У UntrustworthyMailWorker должен быть конструктор от массива MailService (результат вызова processMail первого элемента массива передается на вход processMail второго элемента, и т. д.) и метод getRealMailService, который возвращает ссылку на внутренний экземпляр RealMailService.

2) Spy – шпион, который логгирует о всей почтовой переписке, которая проходит через его руки. Объект конструируется от экземпляра Logger, с помощью которого шпион будет сообщать о всех действиях. Он следит только за объектами класса MailMessage и пишет в логгер следующие сообщения (в выражениях нужно заменить части в фигурных скобках на значения полей почты):
2.1) Если в качестве отправителя или получателя указан "Austin Powers", то нужно написать в лог сообщение с уровнем WARN:
Detected target mail correspondence: from {from} to {to} "{message}"
2.2) Иначе, необходимо написать в лог сообщение с уровнем I*NFO: Usual correspondence: from {from} to {to}*

3) Thief – вор, который ворует самые ценные посылки и игнорирует все остальное. Вор принимает в конструкторе переменную int – минимальную стоимость посылки, которую он будет воровать. Также, в данном классе должен присутствовать метод getStolenValue, который возвращает суммарную стоимость всех посылок, которые он своровал. Воровство происходит следующим образом: вместо посылки, которая пришла вору, он отдает новую, такую же, только с нулевой ценностью и содержимым посылки "stones instead of {content}".

4) Inspector – Инспектор, который следит за запрещенными и украденными посылками и бьет тревогу в виде исключения, если была обнаружена подобная посылка. Если он заметил запрещенную посылку с одним из запрещенных содержимым ("weapons" и "banned substance"), то он бросает IllegalPackageException. Если он находит посылку, состоящую из камней (содержит слово "stones"), то тревога прозвучит в виде StolenPackageException. Оба исключения вы должны объявить самостоятельно в виде непроверяемых исключений.

Все классы должны быть определены как публичные и статические, так как в процессе проверки ваш код будет подставлен во внешний класс, который занимается тестированием и проверкой структуры. Для удобства во внешнем классе объявлено несколько удобных констант и импортировано все содержимое пакета java.util.logging. Для определения, посылкой или письмом является Sendable объект воспользуйтесь оператором instanceof.

```
public static final String AUSTIN_POWERS = "Austin Powers";
public static final String WEAPONS = "weapons";
public static final String BANNED_SUBSTANCE = "banned substance";
```