МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

Санкт-Петербургский национальный исследовательский университет информационных технологий механики и оптики

Мегафакультет трансляционных информационных технологий

Факультет информационных технологий и программирования

Лабораторная работа № 6

По дисциплине «Технологии программирования»

Выполнил студент группы №M3209
Бабурин Тимур

Проверил
Собенников Виктор Леонидович

САНКТ-ПЕТЕРБУРГ
2021

Упражнение 6-1. Обработка исключительных ситуаций.

Цель упражнения: Изучить способы обработки исключительных ситуаций в программах на `Java`.

Описание упражнения: В этом упражнении вы объявите в проекте несколько собственных типов исключений и используете их в методах классов проекта.

1) Объявите в пакете `ru.itmo.exceptions` следующие классы исключений:
   1) `CatalogLoadException` : исключение, выбрасываемое методом `load` интерфейса `CatalogLoader`. Выбрасывается при любых ошибках загрузки каталога.
   2) `ItemAlreadyExistsException` : исключение, выбрасываемое методом `addItem` класса `ItemCatalog`. Выбрасывается в том случае, когда добавляемая позиция уже присутствует в каталоге.
   3) `NegativeQuantityException` : исключение, выбрасываемое методом `remQuantity` класса `Warehouse` (будет создан в последующих упражнениях). Выбрасывается в случае попытки списать со склада количество товара больше имеющегося.

      Примечание: Все эти классы являются проверяемыми исключениями и должны наследовать классу `Exception`.
2) Предусмотрите в методе `load` интерфейса `CatalogLoader` и реализующего его класса `CatalogStubLoader` выброс исключения `CatalogLoadException`. Для класса `CatalogStubLoader` код должен предусматривать ошибки на этапе добавления позиции в каталог:

```
try {
   cat.addItem(item1);
   cat.addItem(item2); }
catch (ItemAlreadyExistsException e) { // TODO Auto-generated
catch block
   e.printStackTrace();
   throw new CatalogLoadException(e);
}
```

3) Предусмотрите в методе `addItem` класса `ItemCatalog` выброс исключения `ItemAlreadyExistsException`. Исключение должно выбрасываться в том случае, когда добавляемая позиция уже есть в каталоге.

4) Исправьте все ошибки, появившиеся в проекте в связи с необходимостью предусмотреть обработку новых исключений.
5) Запустите проект и проверьте его работоспособность. Протестируйте ситуации, в которых могут быть выброшены, добавленные в проект исключения.

Артефакты выполнения упражнения

```java
package ru.billing.exceptions;

public class CatalogLoadException extends Exception{
    public CatalogLoadException(Exception e) { super("CatalogLoadException"); }
}
```

```java
package ru.billing.exceptions;

public class ItemAlreadyExistsException extends Exception{
    public ItemAlreadyExistsException() { super("ItemAlreadyExistsException"); }
}
```

```java
package ru.billing.exceptions;

public class NegativeQuantityException extends Exception {
    public NegativeQuantityException() { super("NegativeQuantityException"); }
}
```

```java
public void addItem(GenericItem item) throws ItemAlreadyExistsException {
    if (catalog.containsKey(item.getID())) {
        throw new ItemAlreadyExistsException();
    }
    catalog.put(item.getID(), item); // Добавляем товар в HashMap
    ALCatalog.add(item); // Добавляем тот же товар в ArrayList
}
```

```java
@Override
public void load(ItemCatalog cat) throws CatalogLoadException {
    GenericItem item1 = new GenericItem( name: "Sony TV", price: 23000, Category.GENERAL);
    FoodItem item2 = new FoodItem( name: "Bread", price: 12, analog: null, new Date(),(short)10);
    try {
        cat.addItem(item1);
        cat.addItem(item2); }
    catch (ItemAlreadyExistsException e) { // TODO Auto-generated catch block
        e.printStackTrace();
        throw new CatalogLoadException(e);
    }
}
```

```
ru.billing.exceptions.ItemAlreadyExistsException Create breakpoint : ItemAlreadyExistsException
    at ru.billing.stocklist.ItemCatalog.addItem(ItemCatalog.java:14)
    at ru.billing.client.CatalogStubLoader.load(CatalogStubLoader.java:15)
    at ru.billing.client.Main.main(Main.java:12)
Exception in thread "main" ru.billing.exceptions.CatalogLoadException Create breakpoint : CatalogLoadException
    at ru.billing.client.CatalogStubLoader.load(CatalogStubLoader.java:18)
    at ru.billing.client.Main.main(Main.java:12)
```

Упражнение 6-2 Синхронизация потоков.

Цель упражнения: Изучить способы синхронизации работы нескольких потоков в многопоточной среде JVM.

Описание упражнения: В этом упражнении вы смоделируете работу банковских операций со счетами.

6) Добавьте в проект новый пакет `sync, в котором буде создавать все классы из этого упражнения`
7) Создайте класс `U1901Bank`, в котором будем имитировать работу банковской операции со счетами и наполните его следующим кодом:
    1) Объявите две `int`-переменные уровня экземпляра класса с именами `intTo` и `intFrom`, которые будут имитировать счет-отправитель и счет-получатель (кредит и дебет). Переменную `intFrom` инициализируйте значением 220.
    2) Создайте метод без возвращаемого значения с именем `calc`. В этом методе будет организована банковская операция по переброске денег между счетами (между переменными `intFrom` и `intTo`). Между снятием денег с одного счета (уменьшением значения переменной `intFrom`) и пополнением другого счета (увеличением переменной `intTo`) будет организована задержка. длительность которой передается в качестве входного параметра. Также в качестве входного параметра передается сумма перевода между счетами. Наполните метод следующим кодом:
        1) Укажите два входных параметра – `intTransaction` (типа int) для передачи суммы и `lngTimeout` (типа `long`) для указания длительности временной задержки.
        2) Для контроля работы метода выведите сообщение, в котором укажите начальные значения переменных `intTo` и `intFrom`, а также имя текущего потока (при помощи метода `Thread.currentThread().getName()`.
        3) Уменьшите значение переменной `intFrom` на значение переменной `intTransaction` и сохраните результат в той же переменной `intFrom`

4) Организуйте временную задержку на `lngTimeout` миллисекунд при помощи метода `Sleep` класса `Thread`. Оберните вызов этого метода в `try/catch`.
5) Увеличьте значение переменной `intTo` на значение переменной `intTransaction` и сохраните результат в той же переменной `intTo`
6) `В завершении выведите имя текущего потока и изменившиеся значения переменных (подобно пункту 2.ii). Рекомендуется в обоих случаях указать какое-нибудь уникальное слово (например before/after) для идентификации вывода.`

8) Создайте класс-наследник от `Thread` с именем `U1901Thread`, в котором будем реализовывать многопоточность для работы с методом `calc` класса `U1901Bank`. Наполните класс следующим кодом:
   1) Объявите следующие переменные уровня экземпляра класса:
      1) `bankWork` типа `U1901Bank`
      2) `intTrans` типа `int`
      3) `lngSleep` типа `long`.
   2) Создайте конструктор, в котором должно быть три параметра для заполнения этих переменных уровня экземпляра класса. Тип параметров должен соответствовать типу переменной, имена параметров можете указать сами. Такой извилистый способ требуется потому, что метод `run()`, который используется для многопоточной работы, не имеет параметров.
   3) Создайте метод `run()`, в коде которого вызовите метод `calc` объекта `bankWork` и передайте в этот метод значения переменных `intTrans` и `lngSleep`.
9) Создайте класс `U1901Main`, который будет вызываться из-под `JVM` для проверки работоспособности многопоточности. В этом классе будет один метод, имя его предлагается определить самостоятельно. В этом методе делаем следующее:
   1) Создать экземпляр класса `U1901Bank` с именем `bankMain` и инициализировать его конструктором.
   2) Создайте экземпляр класса `U1901Thread` с именем `threadOne` и инициализируйте его конструктором со следующими параметрами: `bankMain, 100,2000`.

3) Задайте этому экземпляру уникальное и понятное имя потока при помощи метода `setName` и приоритет `Thread.MAX_PRIORITY` при помощи метода `setPriority`.
4) Запустите поток при помощи метода `start()`
5) Повторите действие пунктов `b,c,d` для экземпляра того же класса, но с именем threadTwo, другим именем потока, конструктором с параметрами: bankMain, 50,300.
6) В качестве контрольного выстрела, в конце метода можно вывести имя текущего потока – `Thread.currentThread().getName()`. Имена всех трех потоков должны быть разными!

10) Запустите класс U1901Main. Вывод должен быть примерно таким -

```
before Thread=thread_100, From=220, To=0
before Thread=thread_50, From=220, To=0
End, main
After Thread=thread_50, From=70, To=50
After Thread=thread_100, From=70, To=150
```

Что здесь не правильно – в выделенной строке сумма значений переменных отличается от суммы в других строках. Это плохо – деньги где-то зависли. Хозяева могут дать по шее.

11) Для исправления сего недостатка надо сделать метод `calc` синхронизированным. После этого вывод будет таким:

```
End, main
before Thread=thread_100, From=220, To=0
After Thread=thread_100, From=120, To=100
before Thread=thread_50, From=120, To=100
After Thread=thread_50, From=70, To=150
```

12) Ура, мы победили – у нас везде `220`!

```java
package sync;

public class U1901Bank {
    public int intTo;
    public int intFrom = 220;

    synchronized void calc(int intTransaction, long lngTimeout) {
        System.out.println("Before-- From:" + intFrom + "    To:" + intTo + "    Thread name:" + Thread.currentThread().getName());
        intFrom = intFrom - intTransaction;
        try {
            Thread.sleep(lngTimeout);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        intTo = intTo + intTransaction;
        System.out.println("After-- From:" + intFrom + "    To:" + intTo + "    Thread name:" + Thread.currentThread().getName());
    }
}
```

```java
package sync;

public class U1901Main {
    public static void main(String args[]) {
        U1901Bank bankMain = new U1901Bank();

        U1901Thread threadOne = new U1901Thread( sumTrans: 100,  sleepTime: 2000, bankMain);
        threadOne.setName("threadOne");
        threadOne.setPriority(Thread.MAX_PRIORITY);
        threadOne.start();

        U1901Thread threadTwo = new U1901Thread( sumTrans: 50,  sleepTime: 300, bankMain);
        threadTwo.setName("threadTwo");
        threadTwo.setPriority(Thread.MAX_PRIORITY);
        threadTwo.start();

        System.out.println(Thread.currentThread().getName());
    }
}
```

```java
package sync;

public class U1901Thread extends Thread{
    U1901Bank bankWork;
    int intTrans;
    long lngSleep;

    U1901Thread (int sumTrans, int sleepTime, U1901Bank bank) {
        this.intTrans = sumTrans;
        this.lngSleep = sleepTime;
        this.bankWork = bank;
    }

    public void run() { bankWork.calc(intTrans, lngSleep); }
}
```

Упражнение 6-3-1 Потоки ввода -вывода.

Цель упражнения: Научиться работать с потоками ввода-вывода.

Описание упражнения: В этом упражнении вы реализуете загрузчик каталога товаров из структурированного текстового файла.

13) Добавьте в пакет `ru.itmo.client` новый класс `CatalogFileLoader`, реализующий интерфейс `CatalogLoader`.

14) Добавьте в класс `CatalogFileLoader` строковое поле, содержащее имя файла со списком товаров и конструктор, принимающий на вход имя этого файла:

```
private String fileName;
public CatalogFileLoader(String fileName) {
   this.fileName = fileName;
}
```

15) Реализуйте метод `load` класса `CatalogFileLoader` следующим образом:

```
File f = new File(fileName);
FileInputStream fis;
String line;
 try {
   fis = new FileInputStream(f);
   Scanner s = new Scanner(fis);

   while(s.hasNextLine()){
   line = s.nextLine();
```

```
    if(line.length()==0) break;
    String[] item_fld = line.split(";");
    String name = item_fld[0];
    float price = Float.parseFloat(item_fld[1]);
    short expires = Short.parseShort(item_fld[2]);
    FoodItem item = new FoodItem(name,price,null, new
Date(),expires);
    cat.addItem(item);
    }
  } catch (FileNotFoundException e) {
    e.printStackTrace();
    throw new CatalogLoadException(e);
  } catch (ItemAlreadyExistsException e) {
    e.printStackTrace();
    throw new CatalogLoadException(e);
  }
}
```

16) Используйте, предоставленную инструктором, версию файла `items.lst` для загрузки каталога товаров.

Артефакты выполнения упражнений:

```java
public class CatalogFileLoader implements CatalogLoader{
    private final String fileName;

    public CatalogFileLoader(String fileName) { this.fileName = fileName; }

    @Override
    public void load(ItemCatalog cat) throws CatalogLoadException {
        File f = new File(fileName);
        FileInputStream fis;
        InputStreamReader read;
        String line;
        try {
            fis = new FileInputStream(f);
            read = new InputStreamReader(fis, charsetName: "WINDOWS-1251");
            Scanner s = new Scanner(read);
            while (s.hasNextLine()) {
                line = s.nextLine();
                if (line.length() == 0)
                    break;
                String[] item_fld = line.split( regex: ";");
                String name = item_fld[0];
                float price = Float.parseFloat(item_fld[1]);
                short expires = Short.parseShort(item_fld[2]);
                FoodItem item = new FoodItem(name, price, analog: null, new Date(), expires);
                cat.addItem(item);
            }
        } catch (FileNotFoundException | ItemAlreadyExistsException | UnsupportedEncodingException e) {
            e.printStackTrace();
            throw new CatalogLoadException(e);
        }
    }
}
```

```java
public class Main {
    public static void main(String[] args) throws ItemAlreadyExistsException, CatalogLoadException {
        CatalogFileLoader aaa = new CatalogFileLoader( fileName: "/home/timurbabs/ITMO/Java/Lab#6/src/items.lst");
        ItemCatalog catalog = new ItemCatalog();
        aaa.load(catalog);
        catalog.printItems();
    }
}
```

```
/usr/lib/jvm/java-15-openjdk/bin/java -javaagent:/home/ssd/IntelliJ IDEA/idea-IU-211.7142.45/lib/idea_rt.jar=43827:/home/ssd/IntelliJ IDEA/idea-IU-211.7142.45/bin -Dfile.encoding=UTF-8 -classpath /home/timurbabs/
ru.billing.stocklist.FoodItem{ID=0, name='Конфеты "Белочка"', price=245.5, analog=null, itemCategory=GENERAL, dateOfIncome=null, expires=120}
ru.billing.stocklist.FoodItem{ID=1, name='Конфеты "Мишка на севере"', price=70.0, analog=null, itemCategory=GENERAL, dateOfIncome=null, expires=120}
ru.billing.stocklist.FoodItem{ID=2, name='Конфеты "Дейл"', price=140.0, analog=null, itemCategory=GENERAL, dateOfIncome=null, expires=170}
ru.billing.stocklist.FoodItem{ID=3, name='Конфеты "Маска"', price=60.0, analog=null, itemCategory=GENERAL, dateOfIncome=null, expires=80}
ru.billing.stocklist.FoodItem{ID=4, name='Шоколад "Аленка"', price=32.0, analog=null, itemCategory=GENERAL, dateOfIncome=null, expires=90}
ru.billing.stocklist.FoodItem{ID=5, name='Шоколад "Вдохновение"', price=45.0, analog=null, itemCategory=GENERAL, dateOfIncome=null, expires=60}
ru.billing.stocklist.FoodItem{ID=6, name='Торт "Орешек"', price=75.5, analog=null, itemCategory=GENERAL, dateOfIncome=null, expires=240}
ru.billing.stocklist.FoodItem{ID=7, name='Торт "Причуда"', price=80.0, analog=null, itemCategory=GENERAL, dateOfIncome=null, expires=260}
ru.billing.stocklist.FoodItem{ID=8, name='Торт "Йогуртовый"', price=220.0, analog=null, itemCategory=GENERAL, dateOfIncome=null, expires=30}
ru.billing.stocklist.FoodItem{ID=9, name='Торт "Ванильный"', price=270.0, analog=null, itemCategory=GENERAL, dateOfIncome=null, expires=7}
ru.billing.stocklist.FoodItem{ID=10, name='Печенье "Юбилейное"', price=50.0, analog=null, itemCategory=GENERAL, dateOfIncome=null, expires=720}
ru.billing.stocklist.FoodItem{ID=11, name='Печенье "Овсяное"', price=45.0, analog=null, itemCategory=GENERAL, dateOfIncome=null, expires=720}
ru.billing.stocklist.FoodItem{ID=12, name='Хлеб "Нарезной"', price=18.0, analog=null, itemCategory=GENERAL, dateOfIncome=null, expires=20}
ru.billing.stocklist.FoodItem{ID=13, name='Хлеб "Бородинский"', price=17.0, analog=null, itemCategory=GENERAL, dateOfIncome=null, expires=30}
ru.billing.stocklist.FoodItem{ID=14, name='Хлеб "Подовый"', price=15.0, analog=null, itemCategory=GENERAL, dateOfIncome=null, expires=18}
ru.billing.stocklist.FoodItem{ID=15, name='Колбаса "Докторская"', price=240.0, analog=null, itemCategory=GENERAL, dateOfIncome=null, expires=14}
ru.billing.stocklist.FoodItem{ID=16, name='Колбаса "Любительская"', price=220.0, analog=null, itemCategory=GENERAL, dateOfIncome=null, expires=14}
ru.billing.stocklist.FoodItem{ID=17, name='Колбаса "Краковская"', price=270.0, analog=null, itemCategory=GENERAL, dateOfIncome=null, expires=30}
ru.billing.stocklist.FoodItem{ID=18, name='Колбаса "Одесская"', price=270.0, analog=null, itemCategory=GENERAL, dateOfIncome=null, expires=35}
ru.billing.stocklist.FoodItem{ID=19, name='Молоко "Буренка"', price=35.0, analog=null, itemCategory=GENERAL, dateOfIncome=null, expires=120}
ru.billing.stocklist.FoodItem{ID=20, name='Молоко "Деревенское"', price=40.0, analog=null, itemCategory=GENERAL, dateOfIncome=null, expires=120}
ru.billing.stocklist.FoodItem{ID=21, name='Картофель "Отборный"', price=30.0, analog=null, itemCategory=GENERAL, dateOfIncome=null, expires=90}
ru.billing.stocklist.FoodItem{ID=22, name='Морковь', price=20.0, analog=null, itemCategory=GENERAL, dateOfIncome=null, expires=30}

Process finished with exit code 0
```