ЛАБОРАТОРНАЯ РАБОТА «ОБРАБОТКА ИСКЛЮЧИТЕЛЬНЫХ СИТУАЦИЙ»

Цель работы:

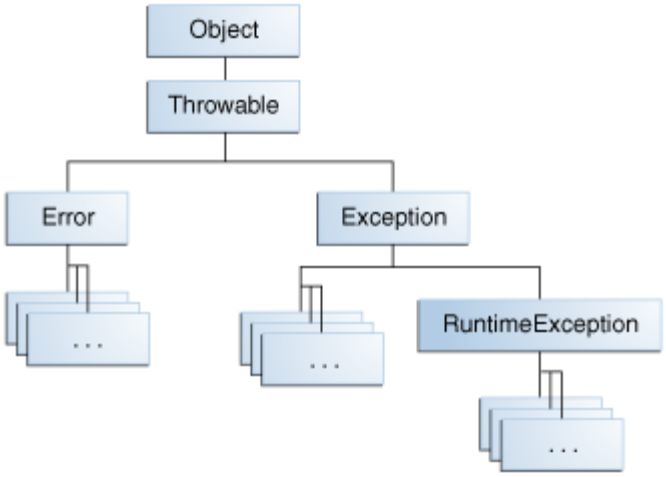
1. ознакомиться с понятием исключительной ситуации;
2. изучить методы обработки исключительных ситуаций;
3. приобрести практические навыки обработки исключительных ситуаций в программах на языке Java;
4. получение навыков описания собственных исключительных ситуаций.

Теоретический материал

Исключениями или исключительными ситуациями (состояниями) называются ошибки, возникшие в программе во время её работы.

Все исключения в Java являются объектами. Поэтому они могут порождаться не только автоматически при возникновении исключительной ситуации, но и создаваться самим разработчиком.

Иерархия классов исключений:



Исключения делятся на несколько классов, но все они имеют общего предка — класс Throwable. Его потомками являются подклассы Exception и Error.

Исключения (Exceptions) являются результатом проблем в программе, которые в принципе решаемы и предсказуемы. Например, произошло деление на ноль в целых числах.

Ошибки (Errors) представляют собой более серьёзные проблемы, которые, согласно спецификации Java, не следует пытаться обрабатывать в собственной программе, поскольку они связаны с проблемами уровня JVM. Например, исключения такого рода возникают, если закончилась память, доступная виртуальной машине. Программа дополнительную память всё равно не сможет обеспечить для JVM.

В Java все исключения делятся на три типа: контролируемые исключения (checked) и неконтролируемые исключения (unchecked), к которым относятся ошибки (Errors) и исключения времени выполнения (RuntimeExceptions, потомок класса Exception).

Контролируемые исключения представляют собой ошибки, которые можно и нужно обрабатывать в программе, к этому типу относятся все потомки класса Exception (но не RuntimeException).

Обработка исключения может быть произведена с помощью операторов try…catch, либо передана внешней части программы. Например, метод может передавать возникшие в нём исключения выше по иерархии вызовов, сам его не обрабатывая.

Неконтролируемые исключения не требуют обязательной обработки, однако, при желании, можно обрабатывать исключения класса RuntimeException.

Откомпилируем и запустим такую программу:

```
class Main {
     public static void main(String[] args) {
         int a = 4;
         System.out.println(a/0);
     }
}
```

В момент запуска на консоль будет выведено следующее сообщение:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero at
Main.main(Main.java:4)
```

Из сообщения виден класс случившегося исключения — ArithmeticException. Это исключение можно обработать:

```
class Main {
     public static void main(String[] args) {
         int a = 4;
         try {
```

```
            System.out.println(a/0);
        } catch (ArithmeticException e) {
            System.out.println("Произошла недопустимая
арифметическая операция");
        }
    }
}
```

Теперь вместо стандартного сообщения об ошибке будет выполняться блок catch, параметром которого является объект **e** соответствующего исключению класса (самому объекту можно давать любое имя, оно потребуется в том случае, если мы пожелаем снова принудительно выбросить это исключение, например, для того, чтобы оно было проверено каким-то ещё обработчиком).

В блок try при этом помещается тот фрагмент программы, где потенциально может возникнуть исключение.

Одному try может соответствовать сразу несколько блоков catch с разными классами исключений.

```
import java.util.Scanner;
class Main {
    public static void main(String[] args) {
     int[] m = {-1,0,1};
        Scanner sc = new Scanner(System.in);
        try {
            int a = sc.nextInt();
            m[a] = 4/a;
            System.out.println(m[a]);
        } catch (ArithmeticException e) {
            System.out.println("Произошла недопустимая арифметическая
операция");
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Обращение по недопустимому индексу
массива");
        }
    }
}
```

Если, запустив представленную программу, пользователь введётся с клавиатуры 1 или 2, то программа отработает без создания каких-либо исключений.

Если пользователь введёт 0, то возникнет исключение класса ArithmeticException, и оно будет обработано первым блоком catch.

Если пользователь введёт 3, то возникнет исключение класса ArrayIndexOutOfBoundsException (выход за приделы массива), и оно будет обработано вторым блоком catch.

Если пользователь введёт нецелое число, например, 3.14, то возникнет исключение класса InputMismatchException (несоответствие типа вводимого значение), и оно будет выброшено в формате стандартной ошибки, поскольку его мы никак не обрабатывали.

Можно, однако, добавить обработчик для класса Exception, поскольку этот класс родительский для всех остальных контролируемых исключений, то он будет перехватывать любые из них (в том числе, и InputMismatchException).

```
import java.util.Scanner;
class Main {
    public static void main(String[] args) {
        int[] m = {-1,0,1};
        int a = 1;
        Scanner sc = new Scanner(System.in);
        try {
            a = sc.nextInt();
            m[a-1] = 4/a;
            System.out.println(m[a]);
        } catch (ArithmeticException e) {
            System.out.println("Произошла недопустимая арифметическая
операция");
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Обращение по недопустимому индексу
массива");
        } catch (Exception e) {
            System.out.println("Произошло ещё какое-то исключение");
        }
    }
}
```

Поскольку исключения построены на иерархии классов и подклассов, то сначала надо пытаться обработать более частные исключения и лишь затем более общие. То есть поставив первым (а не третьим) блок с обработкой исключения класса Exception, мы бы никогда не увидели никаких сообщений об ошибке, кроме «Произошло ещё какое-то исключение» (все исключения перехватились бы сразу этим блоком и не доходили бы до остальных).

Необязательным добавлением к блокам try…catch может быть блок finally. Помещенные в него команды будут выполняться в любом случае, вне зависимости

от того, произошло ли исключение или нет. При том, что при возникновении необработанного исключения оставшаяся после генерации этого исключения часть программы — не выполняется. Например, если исключение возникло в процессе каких-то длительных вычислений, в блоке finally можно показать или сохранить промежуточные результаты.

Порядок выполнения работы

1. **Изучение теоретического материала по лабораторной работе**
2. **Выполнение индивидуального задания работы на базе проекта, созданного для лабораторной работы «Наследование и полиморфизм»**
3. **Оформить отчет**
4. **Подготовить ответы на контрольные вопросы**

Задания на лабораторную работу

В предыдущей лабораторной работе в классах входные параметры методов и конструкторов никак не проверялись. Это может привести к непредсказуемым последствиям. Чтобы избежать подобной ситуации и сообщить пользователю о том, что входные данные некорректные, нужно ввести проверку входных данных с генерацией соответствующего исключения. Но прежде всего, необходимо определить это самое исключение.

Требуется создать указанные в варианте исключения:

Вариант 1). Записная книжка.

1. Конструктор и соответствующий сеттер класса *BirthDay* должны выбрасывать *java.lang.IllegalArgumentException* при попытке передать отрицательное значение возраста.
2. ***Класс*** Person ***должен выбрасывать*** *java.lang.IllegalArgumentException* при попытке установить недопустимый email.
3. Определите объявляемое исключение *TooLateReminderException.* Оно должно расширять класс *Exception* и добавлять 2 поля (с геттерами для них): дата и время события и его напоминания. Классы *BirthDay и Meeting* должны выбрасывать это исключение при попытке установить напоминание позже даты и времени самого события.
4. Определите необъявляемое исключение *DuplicateEventException.* Оно должно расширять класс *RuntimeException* и добавлять одно поле (с геттером для него)

- ссылку на событие, имеющего такие же дату, время и место. Это исключение выбрасывается методом добавления события в классе *Events*, если уже существует **другое**, отличное от добавляемого, событие, но с такими же датой, временем и местом.

5. Переопределите метод, возвращающий массив событий, отсортированный по датам событий.

Вариант 2). Система управления доставкой товара. Создать родительский класс «Заказ» (дата, время, идентификатор) и дочерние классы:

1. Конструктор и соответствующий сеттер класса *Order* должны выбрасывать *java.lang.IllegalArgumentException* при попытке передать отрицательное значение идентификатора заказа.
2. ***Класс InsuredOrder должен выбрасывать*** *java.lang.IllegalArgumentException* при попытке установить отрицательное значение общей суммы застрахованного заказа.
3. Определите объявляемое исключение *TooEarlyReturnDateException.* Оно должно расширять класс *Exception* и добавлять 2 поля (с геттерами для них): дата заказа и значение устанавливаемой даты возврата заказа. Классы *ExpressOrder и InsuredOrder* должны выбрасывать это исключение при попытке установить значение даты возврата раньше, чем дата самого заказа.
4. Определите необъявляемое исключение *DuplicateOrderException.* Оно должно расширять класс *RuntimeException* и добавлять одно поле (с геттером для него) - ссылку на заказ, имеющего такой же идентификатор. Это исключение выбрасывается методом добавления заказа в классе *Orders*, если уже существует **другой**, отличный от добавляемого, заказ, но с таким же идентификатором.
5. Переопределите метод, возвращающий массив заказов, отсортированный по датам.

Вариант 3). Телепрограмма.

1. Конструктор и соответствующий сеттер класса *Children* должны выбрасывать *java.lang.IllegalArgumentException* при попытке передать отрицательные значение минимального и максимального возраста.
2. ***Класс Movie должен выбрасывать*** *java.lang.IllegalArgumentException* при попытке установить отрицательное значение года выпуска фильма.
3. Определите объявляемое исключение *TooEarlyMaxAgeException.* Оно должно расширять класс *Exception* и добавлять 2 поля (с геттерами для них): минимальный возраст и значение устанавливаемого максимального возраста. Класс *Children* должен выбрасывать это исключение при попытке установить значение максимального возраста меньше минимального.
4. Определите необъявляемое исключение *DuplicateProgramException.* Оно должно расширять класс *RuntimeException* и добавлять одно поле (с геттером для него) - ссылку на программу, имеющего такое же время начала. Это исключение выбрасывается методом добавления передачи в классе *DayProgram*, если уже существует **другая**, отличная от добавляемой, передачи, но с таким же

временем.

5. Переопределите метод, возвращающий массив программ, отсортированный по времени начала.

Вариант 4). Гостиница.

1. Конструктор и соответствующий сеттер класса *Room* должны выбрасывать *java.lang.IllegalArgumentException* при попытке передать отрицательные значение в любое из полей класса.
2. ***Класс*** *LuxuryRoom* (комната люкс) ***должен выбрасывать*** *java.lang.IllegalArgumentException* при попытке установить отрицательные значения в поля максимальный и минимальный сроки сдачи.
3. Определите объявляемое исключение *TooEarlyMaxTimeException.* Оно должно расширять класс *Exception* и добавлять 2 поля (с геттерами для них): минимальный срок и значение устанавливаемого максимального срока. Класс *LuxuryRoom* должен выбрасывать это исключение при попытке установить значение максимального возраста меньше минимального.
4. Определите необъявляемое исключение *DuplicateBookingException.* Оно должно расширять класс *RuntimeException* и добавлять одно поле (с геттером для него) - ссылку на номер, который уже забронирован. Это исключение выбрасывается методом бронирования в классах *LuxuryRoom и JuniorSuiteRoom* (комната полулюкс), если номер уже забронирован.
5. Переопределите метод, возвращающий массив номеров, отсортированный по стоимости.

Вариант 5). Реализация готовой продукции.

1. Конструктор и соответствующий сеттер класса *Product (Товар)* должен выбрасывать *java.lang.IllegalArgumentException* при попытке передать отрицательные значение в любое из полей класса: идентификатор, код, цена.
2. ***Классы*** *FragileProduct* (хрупкий товар) и *PerishableProduct* (скоропортящийся товар) ***должен выбрасывать*** *java.lang.IllegalArgumentException* при попытке установить отрицательные значения в поля коэффициент хрупкости и максимальный срок хранения.
3. Определите объявляемое исключение *TooLateDeliveryTimeException.* Оно должно расширять класс *Exception* и добавлять 2 поля (с геттерами для них): максимальный срок хранения товара и значение устанавливаемой даты доставки товара. Класс *PerishableProduct* должен выбрасывать это исключение при попытке установить значение даты доставки, превышающей срок хранения товара.
4. Определите необъявляемое исключение *DuplicateWriting_OffException.* Оно должно расширять класс *RuntimeException* и добавлять одно поле (с геттером для него) - ссылку на, товар, который уже списан. Это исключение выбрасывается методом списания в классах *PerishableProduct и FragileProduct*, если товар уже списан.

5. Переопределите метод, возвращающий массив товаров, отсортированный по стоимости.

Вариант 6). Учет успеваемости студентов ВУЗА.

1. Конструктор и соответствующий сеттер класса *Student (Студент)* должен выбрасывать *java.lang.IllegalArgumentException* при попытке установить недопустимое значение оценки (<2 или >5) или передать пустую дисциплину.
2. ***Классы*** *DistanceLearning* (Заочное обучение) и *TargetEducation* (целевое обучение) ***должен выбрасывать*** *java.lang.IllegalArgumentException* при попытке установить отрицательные значения в поле суммы обучения.
3. Определите объявляемое исключение *IncorrectPaymentSumException.* Оно должно расширять класс *Exception* и добавлять 2 поля (с геттерами для них): сумма за обучение и значение устанавливаемой суммы оплаты. Классы должны выбрасывать это исключение при попытке внести сумму оплаты за обучение.
4. Определите необъявляемое исключение *DuplicateExpelException.* Оно должно расширять класс *RuntimeException* и добавлять одно поле (с геттером для него) - ссылку на студента, которого уже отчислили. Это исключение выбрасывается методом отчисления студента в соответствующих классах, в случае если он уже отчислен.
5. Переопределите метод, возвращающий массив студентов, отсортированный по фамилиям.

Вариант 7). Автоматизация работы факультета.

1. Конструктор и соответствующий сеттер класса *Faculty (Факульет)* должен выбрасывать *java.lang.IllegalArgumentException* при попытке передать некорректный номер телефона (правильный номер состоит из 6 цифр).
2. ***Классы*** *FragileProduct* (хрупкий товар) и *PerishableProduct* (скоропортящийся товар) ***должен выбрасывать*** *java.lang.IllegalArgumentException* при попытке установить отрицательные значения в поля коэффициент хрупкости и максимальный срок хранения.
3. Определите объявляемое исключение *IncorrectPositionException.* Оно должно расширять класс *Exception* и добавлять 2 поля (с геттерами для них): возможные наименования должностей на кафедре (заведующий, профессор, доцент, старший преподаватель, ассистент, лаборант) и значение устанавливаемой должности. Класс *DepartmentTeachers* должен выбрасывать это исключение при попытке установить недопустимое значение должности преподавателя.
4. Определите необъявляемое исключение *DuplicateHiringException.* Оно должно расширять класс *RuntimeException* и добавлять одно поле (с геттером для него) - ссылку на, сотрудника, ФИО которого ранее уже приняли на работу. Это

исключение выбрасывается методом списания в классах *DepartmentTeachers и FacultyStaff*, если добавляемый сотрудник уже существует.

5. Переопределите метод, возвращающий массив преподавателей, отсортированный по стажу работы по убыванию.

Вариант 8). Продажа товаров супермаркета.

1. Конструктор и соответствующий сеттер класса *ProductDepartment (Отделы товаров)* должен выбрасывать *java.lang.IllegalArgumentException* при попытке передать отрицательные значение в поля: розничная и закупочная цена.
2. ***Класс*** *Fruits* (фрукты) ***должен выбрасывать*** *java.lang.IllegalArgumentException* при попытке установить отрицательные значения в поля максимальный срок и температура.
3. Определите объявляемое исключение *TooLowPriceException.* Оно должно расширять класс *Exception* и добавлять 2 поля (с геттерами для них): закупочная цена на товар и значение устанавливаемой розничной цены товара. Классы *Toys* и *BigScaleProducts* должны выбрасывать это исключение при попытке установить значение розничной цены (и в случае скидки) ниже закупочной.
4. Определите необъявляемое исключение *DuplicateWriting_OffException.* Оно должно расширять класс *RuntimeException* и добавлять одно поле (с геттером для него) - ссылку на, товар, который уже списан. Это исключение выбрасывается методом списания в классе *Fruits*, если товар уже списан.
5. Переопределите метод, возвращающий массив товаров, отсортированный по стоимости.

Вариант 9). Учет военного состава.

1. Конструктор и соответствующий сеттер класса *Serviceman (Военнослужащий)* должен выбрасывать *java.lang.IllegalArgumentException* при попытке передать пустые значения в поля: фамилия и звание.
2. ***Класс*** *MilitaryAdministration* (Органы военного управления) ***должен выбрасывать*** *java.lang.IllegalArgumentException* при попытке установить отрицательные значения в поля выслуга лет и сумма надбавки.
3. Определите объявляемое исключение *TooHighAwardAmountException.* Оно должно расширять класс *Exception* и добавлять 2 поля (с геттерами для них): премия и значение устанавливаемой суммы надбавки. Классы *MilitaryAdministration* и *ContractMilitaryService* (Военная служба по контракту) должны выбрасывать это исключение при попытке установить сумму надбавки выше премии.
4. Определите необъявляемое исключение *DuplicateContractException.* Оно должно расширять класс *RuntimeException* и добавлять одно поле (с геттером для него) - ссылку на военнослужащего по контракту, у которого уже есть действующий контракт. Это исключение выбрасывается при заполнении данных о контракте в классе *ContractMilitaryService*, если новый контракт покрывает действующий контракт.
5. Переопределите метод, возвращающий массив военнослужащих,

отсортированный по фамилиям.

Вариант 10). Учет литературы.

1. Конструктор и соответствующий сеттер класса *Literature (Литература)* должен выбрасывать *java.lang.IllegalArgumentException* при попытке передать пустые значения в поля: название и автор.
2. ***Класс*** *ReferenceBook* (Справочник) ***должен выбрасывать*** *java.lang.IllegalArgumentException* при попытке установить отрицательные значения в поля том и часть.
3. Определите объявляемое исключение *DuplicateSubscriptionException.* Оно должно расширять класс *Exception* и добавлять 2 поля (с геттерами для них): период подписки и значение устанавливаемого нового периода. Класс *Periodicals* (Периодика) должен выбрасывать исключение при попытке повторно оформить подписку на тот же период.
4. Определите необъявляемое исключение *DuplicateWriting_OffException.* Оно должно расширять класс *RuntimeException* и добавлять одно поле (с геттером для него) - ссылку на литературу, которая уже была ранее списана. Это исключение выбрасывается соответствующими классами при попытке повторно списать какую-либо литературу.
5. Переопределите метод, возвращающий массив книг, отсортированный по названиям.

Вариант 11). Учет продажи путевок.

1. Конструктор и соответствующий сеттер класса *Voucher (Путевка)* должен выбрасывать *java.lang.IllegalArgumentException* при попытке передать отрицательные значения в поля: количество человек и цена.
2. ***Класс*** *ChildrenVoucher* (Справочник) ***должен выбрасывать*** *java.lang.IllegalArgumentException* при попытке установить отрицательное значение в поле возраст ребенка.
3. Определите объявляемое исключение *InvalidExitDateException.* Оно должно расширять класс *Exception* и добавлять 2 поля (с геттерами для них): дата заезда и значение устанавливаемой даты выезда. Класс *Voucher* (Путевка) должен выбрасывать исключение при попытке установить дату выезда, предшествующую дате заезда.
4. Определите необъявляемое исключение *DuplicateVoucherReturningException.* Оно должно расширять класс *RuntimeException* и добавлять одно поле (с геттером для него) - ссылку на путевку, которая уже была ранее возвращена. Это исключение выбрасывается соответствующими классами при попытке повторно вернуть путевку.
5. Переопределите метод, возвращающий массив путевок, отсортированный по фамилиям клиентов.

Вариант 12). Учет выполненных работ станции техобслуживания.

1. Конструктор и соответствующий сеттер класса *Servicing (Техобслуживание)* должен выбрасывать *java.lang.IllegalArgumentException* при попытке передать отрицательное значение в поле: сумма ремонта.

2. ***Класс*** *CarsPreventiveMaintenance* (Планово-предупредительный осмотр для легкового автотранспорта) ***должен выбрасывать*** *java.lang.IllegalArgumentException* при попытке установить отрицательные значения в поля: пробег и год проведения issuancedate.
3. Определите объявляемое исключение *InvalidIssuanceDateException.* Оно должно расширять класс *Exception* и добавлять 2 поля (с геттерами для них): дата поступления и значение устанавливаемой даты выдачи. Класс *Servicing* (Техобслуживание) должен выбрасывать исключение при попытке установить дату выдачи, предшествующую дате поступления.
4. Определите необъявляемое исключение *DuplicateDiagnosticException.* Оно должно расширять класс *RuntimeException* и добавлять одно поле (с геттером для него) - ссылку на транспортное средство, для которого уже была в текущем месяце проведена диагностика. Это исключение выбрасывается соответствующими классами при попытке повторно за месяц провести диагностику транспортного средства.
5. Переопределите метод, возвращающий массив автомобилей на ремонте, отсортированный по дате поступления.

Вариант 13). Медицинское обслуживание пациентов

1. Конструктор и соответствующие сеттеры класса *MedicalServicing (Медобслуживание)* должен выбрасывать *java.lang.IllegalArgumentException* при попытке передать пустые значения в поля: фамилия пациента и номер полиса.
2. ***Класс*** *PreventiveMaintenance* (Планово-предупредительный осмотр) ***должен выбрасывать*** *java.lang.IllegalArgumentException* при попытке установить отрицательные значения в поля: год проведения и период действия.
3. Определите объявляемое исключение *InvalidAppointmentTimeException.* Оно должно расширять класс *Exception* и добавлять 2 поля (с геттерами для них): дата и время приема у врача и значение устанавливаемой даты и времени приема для пациента. Класс *MedicalServicing* (Медобслуживание) должен выбрасывать исключение при попытке установить время приема пациенту, на которое у врача уже есть запись.
4. Определите необъявляемое исключение *DuplicateVaccinationException.* Оно должно расширять класс *RuntimeException* и добавлять одно поле (с геттером для него) - ссылку на человека, для которого уже была в текущем периоде проведена соответствующая вакцинация. Это исключение выбрасывается соответствующими классами при попытке провести вакцинацию одной и той же вакциной до истечения периода ее действия.
5. Переопределите метод, возвращающий массив пациентов, отсортированный по убыванию даты осмотра.

Вариант 14). Библиотека.

1. Конструктор и соответствующие сеттеры класса *Literature (Литература)* должен выбрасывать *java.lang.IllegalArgumentException* при попытке передать

отрицательные значения в поля: год издания и количество страниц.

2. ***Класс*** *ReadingRoom* (Читательский зал) ***должен выбрасывать*** *java.lang.IllegalArgumentException* при попытке установить отрицательные значения в поля: количество источников, этаж, кабинет.
3. Определите объявляемое исключение *InvalidIssuanceDateException.* Оно должно расширять класс *Exception* и добавлять 2 поля (с геттерами для них): текущая дата и значение устанавливаемой даты выдачи литературы. Класс *LiteratureIssuance* (Выдача литературы) должен выбрасывать исключение при попытке установить дату выдачи, предшествующую текущей дате.
4. Определите необъявляемое исключение *DuplicateDeactivationException.* Оно должно расширять класс *RuntimeException* и добавлять одно поле (с геттером для него) - ссылку на читателя (или книгу), которые ранее уже были деактивированы. Это исключение выбрасывается соответствующими классами при попытке деактивировать повторно один и тот же объект.
5. Переопределите метод, возвращающий массив читателей, отсортированный по фамилиям.

Вариант 15). Продажа автомобилей.

1. Конструктор и соответствующий сеттер класса *Automobile (Автомобиль)* должен выбрасывать *java.lang.IllegalArgumentException* при попытке передать отрицательные значение в поля: розничная и закупочная цена.
2. ***Классы автомобилей должны выбрасывать*** *java.lang.IllegalArgumentException* при попытке установить отрицательные значения в поле максимальная допустимая масса груза.
3. Определите объявляемое исключение *TooLowPriceException.* Оно должно расширять класс *Exception* и добавлять 2 поля (с геттерами для них): закупочная цена на автомобиль и значение устанавливаемой розничной цены. Классы должны выбрасывать это исключение при попытке установить значение розничной цены ниже закупочной.
4. Определите необъявляемое исключение *DuplicateUtilizationException.* Оно должно расширять класс *RuntimeException* и добавлять одно поле (с геттером для него) - ссылку на автомобиль, который уже утилизирован. Это исключение выбрасывается методом утилизации в соответствующих классах, если автомобиль уже был утилизирован.
5. Переопределите метод, возвращающий массив автомобилей салона, упорядоченный по стоимости.

Вариант 16). Учет продаж через Интернет-магазин.

1. Конструктор и соответствующий сеттер класса *Product (Товар)* должен выбрасывать *java.lang.IllegalArgumentException* при попытке передать отрицательные значение в поля: розничная и закупочная цена.
2. ***Классы товаров должны выбрасывать*** *java.lang.IllegalArgumentException* при попытке установить отрицательные значения в поля, определяющие размеры мебели.
3. Определите объявляемое исключение *TooLowPriceException.* Оно должно расширять класс *Exception* и добавлять 2 поля (с геттерами для них): закупочная цена на товар и значение устанавливаемой розничной цены.

Классы должны выбрасывать это исключение при попытке установить значение розничной цены ниже закупочной.

4. Определите необъявляемое исключение *DuplicateUtilizationException.* Оно должно расширять класс *RuntimeException* и добавлять одно поле (с геттером для него) - ссылку на товар, который уже утилизирован. Это исключение выбрасывается методом утилизации в соответствующих классах, если товар уже был утилизирован.
5. Переопределите метод, возвращающий массив товаров, упорядоченный по дате поступления.

Контрольные вопросы

1. Что такое исключение?
2. Основные принципы обработки исключений.
3. Что такое класс Exception?
4. Что такое блок try/catch/finally?
5. Какой класс является базовым для всех исключительных ситуаций?
6. Что такое контролируемые и неконтролируемые исключения?
7. Как выполняется обработка исключений во вложенных блоках try ..catch … finally
8. Для чего используется ключевое слово throws?
9. Как реализовать общую обработку для нескольких видов исключений – приведите несколько способов.
10. Для чего используется зарезервированное слово throw в Java?
11. Чем отличаются исключения, производные от класса RuntimeException от всех остальных?
12. Есть ли ошибка в следующем коде

```
try {

} finally {

}
```

13. Какие типы исключений будут обрабатываться следующей конструкцией?

```
catch (Exception e) {

}
```

14. Какая ошибка в следующем фрагменте?

```
try {

} catch (Exception e) {

} catch (ArithmeticException a) {

}
```

15. Приведите пример собственного класса исключительной ситуации (отличного от примеров в лекции)