Plan

Русский текст смотри ниже

Plan of lesson

1. Homework explanation
2. Enum - enum type in Java
3. Handling exceptional situations. Exception

План на урок

1. Разбор домашнего задания
2. Enum - перечисляемый тип в Java
3. Обработка исключительных ситуаций. Exception

Theory

Русский текст смотри ниже

1. An enumeration type is a special data type that allows a variable to be a set of predefined constants. The variable must be equal to one of its predefined values. Because they are constants, the field names of the enumerated type are written in capital letters.
2. In the Java programming language, the type of an enumeration is defined using the enum keyword. For example, you can specify the enumeration type for days of the week like this: public enum Day { SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY } All enums implicitly extend java.lang.Enum. Here is a link to the official Oracle tutorial on enumerated types: Enum
3. An error in a method interrupts its work at the point of origin of the error, and destroys the entire stack of functions. At the same time, Java creates and “forwards” a special object characterizing the error. All such objects inherit the Throwable class.
4. All Throwable descendants are divided into two categories: Error and Exception. Error are errors with which nothing can be done. For example, lack of memory. The second category, Exception - exceptional situations caused by an incorrect state of the program.
5. Errors are of two types: checked (checked) and unchecked (unchecked). The first type the compiler “sees” and forces the programmer to somehow react to them. How to respond to unverifiable errors (and whether to respond at all) is up to the developer to decide.
6. Error handling is carried out using the try-catch-finally construct. In try blocks we indicate what to do if everything is fine. We handle errors in the catch block. The finally block is executed in any case.
7. We can create our own errors and “throw” them using the throw keyword. If we want to create a checked exception, then we can extend the Exception class. If unchecked, then we

inherit the RuntimeException class.
8. If we encounter a checked exception, but do not want to handle it in this method using try-catch, and decide to throw this exception to the calling method, then we can use the throws keyword in the method signature.

1. Тип перечисления — это специальный тип данных, который позволяет переменной быть набором предопределенных констант. Переменная должна быть равна одному из предопределенных для нее значений. Поскольку они являются константами, имена полей перечисляемого типа пишутся большими буквами.
2. В языке программирования Java тип перечисления определяется с помощью ключевого слова enum. Например, вы можете указать тип перечисления дней недели следующим образом: public enum Day { SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY } Все перечисления неявно расширяют java.lang.Enum. Вот ссылка на официальный Oracle туториал по перечисляемым типам: Enum
3. Ошибка в методе, прерывает его работы в точке происхождения ошибки, и разрушает весь стек функций.При этом Java создает и "пробрасывает" специальный объект характеризующий ошибку. Все такие объекты наследуют классу Throwable.
4. Все наследники Throwable делятся на две категории: Error и Exception. Error - это ошибки с которыми ничего сделать нельзя. Например нехватка памяти. Вторая категория, Exception - исключительные ситуации вызванные некорректным состоянием программы.
5. Ошибки бывают двух типов: checked (проверяемые) и unchecked (непроверяемые). Первый тип компилятор "видит" и заставляет программиста как-то на них отреагировать. Как реагировать на непроверяемые ошибки (и реагировать ли вообще), решает сам разработчик.
6. Обработка ошибок осуществляется при помощи конструкции try-catch-finally. В блоки try мы указываем, что делать если все нормально. В блоке catch обрабатываем ошибки. Блок finally выполняется в любой случае.
7. Мы можем создавать свои ошибки и "бросать" их при помощи ключего слова throw. Если мы хотим создать проверяемое исключение, то можно наследовать классу Exception. Если непроверяемое, то наследуем классу RuntimeException.
8. Если мы столкнулись с проверяемым исключением, но не хотим его обрабатывать в данном методе при помощи try-catch, и решили пробросить это исключение в вызывающий метод, то можно воспользоваться ключевым словом throws в сигнатуре метода.

Homework

Русский текст смотри ниже

Task 1

See TODO in the archive with the project Company in Slack.

Задача 1

Смотри TODO в архиве с проектом Company в Slack.

Code

code/src/homeworks/GarageImpl.java

```
package ait.cars.dao;

import ait.cars.model.Car;

import java.util.Arrays;
import java.util.function.Predicate;

public class GarageImpl implements Garage {
    private Car[] cars;
    private int size;

    public GarageImpl(int capacity) {
        cars = new Car[capacity];
    }

    @Override
    public boolean addCar(Car car) {
        if (car == null || size == cars.length || findCarByRegNumber(car.ge
            return false;
        }
        cars[size++] = car;
        return true;
    }

    @Override
    public Car removeCar(String regNumber) {
        for (int i = 0; i < size; i++) {
            if (regNumber.equals(cars[i].getRegNumber())) {
                Car temp = cars[i];
//                cars[i] = cars[--size];
                System.arraycopy(cars, i + 1, cars, i, size - i - 1);
                cars[--size] = null;
                return temp;
```

```
            }
        }
        return null;
    }

    @Override
    public Car findCarByRegNumber(String regNumber) {
        for (int i = 0; i < size; i++) {
            if (cars[i].getRegNumber().equals(regNumber)) {
                return cars[i];
            }
        }
        return null;
    }

    @Override
    public Car[] findCarsByModel(String model) {
        return findCarsByPredicate(c -> model.equals(c.getModel()));
    }

    @Override
    public Car[] findCarsByCompany(String company) {
        return findCarsByPredicate(c -> company.equals(c.getCompany()));
    }

    @Override
    public Car[] findCarsByEngine(double min, double max) {
        return findCarsByPredicate(c -> c.getEngine() >= min && c.getEngine
    }

    @Override
    public Car[] findCarsByColor(String color) {
        return findCarsByPredicate(c -> color.equals(c.getColor()));
//        return findCarsByPredicate(c -> c.getColor().equals(color));
    }

    private Car[] findCarsByPredicate(Predicate<Car> predicate) {
        Car[] res = new Car[size];
        int j = 0;
        for (int i = 0; i < size; i++) {
            if (predicate.test(cars[i])) {
                res[j++] = cars[i];
            }
```

```
            }
            return Arrays.copyOf(res, j);
        }

    }
```

code/src/Season.java

```
    package ait.enums.model;

    public enum Season {
        WINTER, SPRING, SUMMER, AUTUMN;
    }
```

code/src/SeasonAppl.java

```
    package ait.enums;

    import ait.enums.model.Season;

    public class SeasonAppl {
        public static void main(String[] args) {
            Season season = Season.AUTUMN;
            System.out.println(season);
            System.out.println(Season.SUMMER);
            System.out.println(season.name());
            System.out.println(Season.WINTER.name());
            season = Season.valueOf("SPRING");
            System.out.println(season);
            System.out.println("===== values =====");
            Season[] seasons = Season.values();
            for (int i = 0; i < seasons.length; i++) {
                System.out.println(seasons[i]);
            }
            System.out.println("===== ordinal =====");
            System.out.println(season.name());
            System.out.println(season.ordinal());
            System.out.println(Season.AUTUMN.ordinal());
        }
    }
```

code/src/Month.java

```java
package ait.enums.model;

public enum Month {
    JAN(31), FEB(28), MAR(31), APR, MAY(31), JUN(30), JUL(31), AUG(31),
    SEP(30), OCT(31), NOV(30), DEC(31);

    private final int days;

    Month() {
        days = 30;
    }
    Month(int days) {
        this.days = days;
    }

    public int getDays() {
        return days;
    }


    public Month plustMonth(int quantity) {
        Month[] values = values();
        int index = ordinal();
        index = index + quantity;
        return values[index % values.length];
    }

    public static String getName(int position) {
        Month[] values = values();
        return values[(position - 1) % values.length].name();
    }
}
```

code/src/MonthAppl.java

```java
package ait.enums;

import ait.enums.model.Month;

public class MonthAppl {
    public static void main(String[] args) {
        Month month = Month.APR;
        System.out.println(month);
```

```
            System.out.println(month.plustMonth(4));
            System.out.println(Month.OCT.plustMonth(5));
            System.out.println(Month.getName(13));
            System.out.println(Month.JUL.getDays());
        }
    }
```

code/src/ExceptionAppl.java

```
    package ait.exception;

    public class ExceptionAppl {
        public static void main(String[] args) {
            int a = 0;
            int b = 0;
            try {
                int x = solution(a, b);
                System.out.println("Solution = " + x);
            } catch (ArithmeticException e) {
    //            e.printStackTrace();
                System.out.println("Solution any number");
                return;
            } catch (IllegalArgumentException e) {
                System.out.println("No solution");
                return;
            } finally {
                System.out.println("Bye, bye");
            }
            System.out.println("Math is great!");
        }

        public static int solution(int a, int b) {
            if (a == 0 && b != 0) {
                throw new IllegalArgumentException();
            }
            if (a == 0 && b == 0) {
                throw new ArithmeticException();
            }
            int res = b / a;
            return res;
        }
    }
```