

Занятие 7: «Классы, исключения и ФЛЕКС»

Инфохимия, 1 курс, зима

презентацию подготов... На
Эту. Часть.

А что от нас хотят?

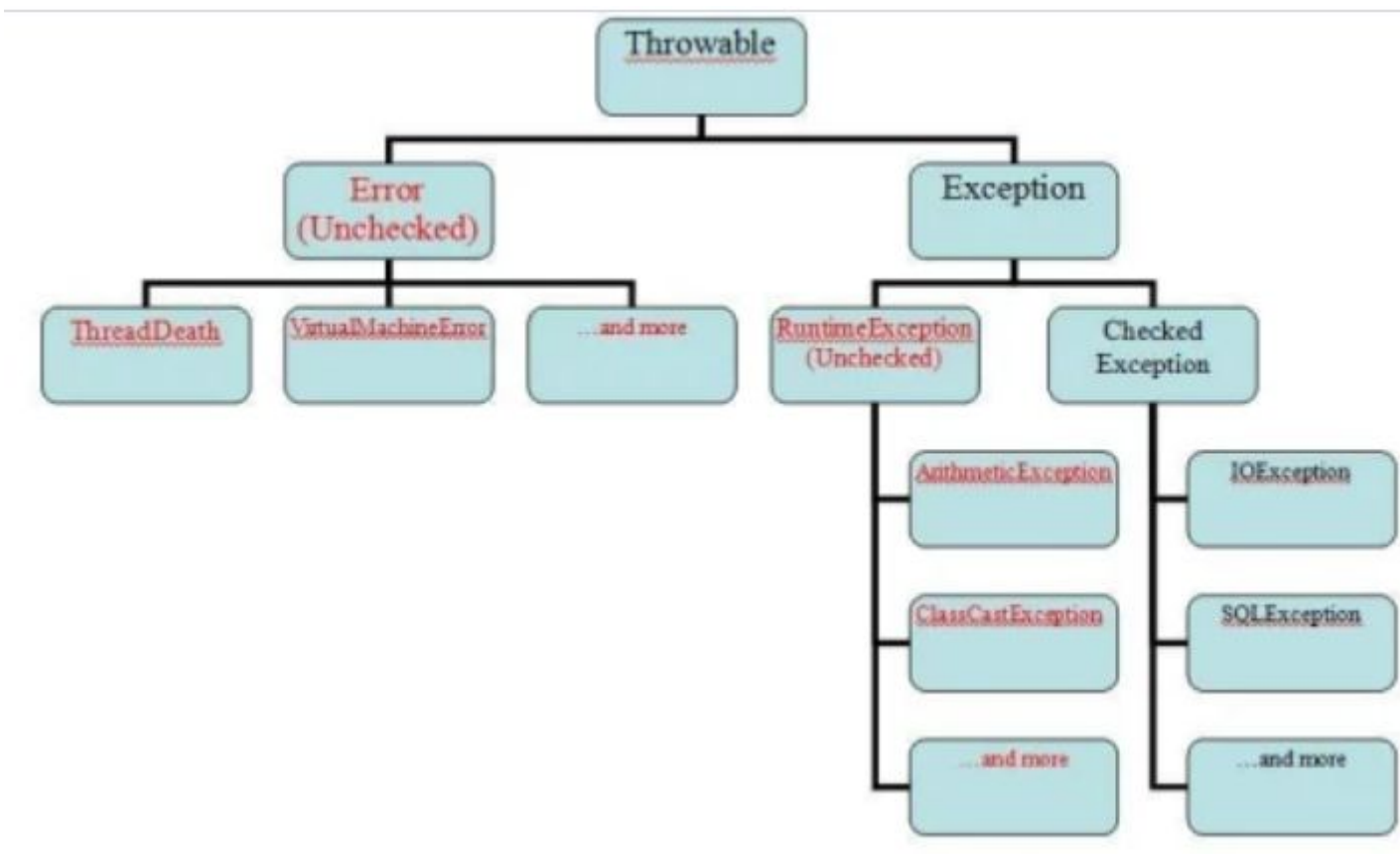
1. Обработка исключительных ситуаций, три типа исключений.
2. Вложенные, локальные и анонимные классы.
3. Механизм рефлексии (reflection) в Java. Класс `Class`.

PLAN УРОКА

- Что такое исключения
- Как обрабатывать исключения
- Вложенные классы
- Локальные классы
- Анонимные классы
- РеФЛЕКСия?
- Класс Класс Класс Класс Класс

Что такое эти ваши исключения?

возникновение ошибок и непредвиденных ситуаций при выполнении программы



Как обрабатывать исключения

try, catch, finally, throws

try{}catch, try{}catch{}finally, try{}finally{}

try – определяет блок кода, в котором может произойти исключение;

catch – определяет блок кода, в котором происходит обработка исключения;

finally – определяет блок кода, который является необязательным, но при его наличии выполняется в любом случае независимо от результатов выполнения блока try.

throw – используется для возбуждения исключения;

throws – используется в сигнатуре методов для предупреждения, о том что метод может выбросить исключение.

```
public String input() throws MyException {  
    BufferedReader reader = new BufferedReader(new  
        InputStreamReader(System.in));  
    String s = null;  
    try{  
        s = reader.readLine();  
    } catch(IOException e){  
        System.out.println(e.getMessage());  
    } finally {  
        try{  
            reader.close();  
        } catch(IOException e){  
            System.out.println(e.getMessage());  
        }  
    }  
    if(s.equals("")){  
        throw new MyException("String can not be empty!");  
    }  
    return s;  
}
```

Виды классов (ОЧЕНЬ КРАТКОOOOOOOOOOOOOOOOOOOOO OOOOOOOOOOO)

Вложенные внутренние классы – нестатические классы внутри внешнего класса.

Вложенные статические классы – статические классы внутри внешнего класса.

Локальные классы Java – классы внутри методов.

Анонимные Java классы – классы, которые создаются на ходу.

Вложенные нестатические классы

```
class Human{
double height, weight;
String name;
public Human(double h, double w, String name){
height = h;
weight = w;
this.name = name;
}
Head head_from_Valya = new Head(25);

private class Head{
double weight;
private Head(double w){
this.weight = w;
}
}
}
```

Особенности

1. Существуют только у объектов, поэтому для их создания нужен объект, то есть без человека не может быть головы.
2. Внутри такого класса не может быть статических переменных из-за тесной связи нестатического вложенного класса с внешним классом.
3. У класса полный доступ ко всем приватным полям внешнего класса, как и наоборот.
4. Можно получить ссылку на объект внешнего класса. Human.this – ссылка на человека, this – ссылка на голову.

Вложенные статические

```
class Place{
double P, S;
String name, LevelOfLight;
int GroundLevel;
public Place(double P, double S, String name, String LOL, int GL){
this.P = P;
this.S = S;
this.name = name;
this.LevelOfLight = LOL;
this.GroundLevel = GL;
}
public void Broken(String consequences){
System.out.println(Scene.FirstWord(this.name) + " was destroyed and " + consequences + ".");
}

static class Store extends Place{

public Store(double P, double S, String name, String LOL, int GL){
super(P, S, name, LOL, GL);
}
}

static class MainRoom extends Store{
public MainRoom(double P, double S, String name, String LOL, int GL){
super(P, S, name, LOL, GL);
}
}
```

Делаем экземпляр

```
Place.Store store = new Place.Store(200, 200, "the store", "nice", 1);
Place.Store.MainRoom mr = new Place.Store.MainRoom(50, 100, "the main room", "well", 1);
Place.Store.Storage st = new Place.Store.Storage(25, 50, "the storage", "good", 1);
Place.Tunnel tun = new Place.Tunnel(120, 75, "the tunnel", "bad", 1);
Place.Garage gar = new Place.Garage(60, 50, "the garage", "normal", -1);
Place.Street str = new Place.Street(9999999, 999999, "the street", "great", 1);
Place.Tunnel underground_tun = new Place.Tunnel(240, 200, "the underground tunnel", "bad", -2);
Place.PoliceStation ps = new Place.PoliceStation(500, 600, "the police station", "nice", 1);
```

Плюсы такого подхода:

1. Количество классов уменьшилось.
2. Все классы внутри их класса-родителя. Мы способны проследживать всю иерархию без открытия каждого класса отдельно.
3. Мы можем обратиться к классу Building, а IDE уже будет подсказывать весь список всех подклассов данного класса. Это будет упрощать поиск нужных классов и показывать всю картину более цело.

Локальные классы

1. Локальные классы способны работать только с `final` переменными метода.
2. Локальные классы нельзя объявлять с модификаторами доступа.
3. Локальные классы обладают доступом к переменным метода.

```
public class Person {
    private String name, street, house;

    public Person(String name, String street, String house) {
        this.name = name;
        this.street = street;
        this.house = house;
    }

    private interface AddressContainer {
        String getStreet();
        String getHouse();
    }

    public AddressContainer getAddressContainer() {
        class PersonAddressContainer implements AddressContainer {
            final String street = Person.this.street, house = Person.this.house;

            @Override
            public String getStreet() {
                return this.street;
            }

            @Override
            public String getHouse() {
                return this.house;
            }
        }

        return new PersonAddressContainer();
    }
}
```

Анонимные классы

```
public class Animal {
    public void meow() {
        System.out.println("Meow!");
    }

    public static void main(String[] args) {
        Animal anonTiger = new Animal() {
            @Override
            public void meow() {
                System.out.println("Raaar!");
            }
        };

        Animal notAnonTiger = new Animal().new Tiger();

        anonTiger.meow(); // будет выведено Raaar!
        notAnonTiger.meow(); // будет выведено Raaar!
    }

    private class Tiger extends Animal {
        @Override
        public void meow() {
            System.out.println("Raaar!");
        }
    }
}
```

Используем, когда

- тело класса является очень коротким;
- нужен только один экземпляр класса;
- класс используется в месте его создания или сразу после него;
- имя класса не важно и не облегчает понимание кода.

РеФЛЕКСия!

Рефлексия (от позднелат. *reflexio* — обращение назад) — это механизм исследования данных о программе во время её выполнения. Рефлексия позволяет исследовать информацию о полях, методах и конструкторах классов.

За че

- Узнать/определить класс объекта;
- Получить информацию о модификаторах класса, полях, методах, константах, конструкторах и суперклассах;
- Выяснить, какие методы принадлежат реализуемому интерфейсу/интерфейсам;
- Создать экземпляр класса, причем имя класса неизвестно до момента выполнения программы;
- Получить и установить значение поля объекта по имени;
- Вызвать метод объекта по имени.

```
public class MyClass {  
    private int number;  
    private String name = "default";  
    //    public MyClass(int number, String name) {  
    //        this.number = number;  
    //        this.name = name;  
    //    }  
    public int getNumber() {  
        return number;  
    }  
    public void setNumber(int number) {  
        this.number = number;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    private void printData(){  
        System.out.println(number + name);  
    }  
}
```

```
public static void main(String[] args) {  
    MyClass myClass = new MyClass();  
    int number = myClass.getNumber();  
    String name = null; //no getter =(  
    System.out.println(number + name); //output 0null  
    try {  
        Field field = myClass.getClass().getDeclaredField("name");  
        field.setAccessible(true);  
        name = (String) field.get(myClass);  
    } catch (NoSuchFieldException | IllegalAccessException e) {  
        e.printStackTrace();  
    }  
    System.out.println(number + name); //output 0default  
}
```

КЛАСС?!

Instances of the class `Class` represent classes and interfaces in a running Java application.

Имеем...

1. Обработка исключительных ситуаций, три типа исключений.
2. Вложенные, локальные и анонимные классы.
3. Механизм рефлексии (reflection) в Java. Класс `Class`.



СПАСИБО ЗА ВНИМАНИЕ!!!