Języki i Techniki Programowania

Laboratorium nr 3

Metody i atrybuty statyczne

- **Pola (atrybuty) statyczne**
- Pola klasowe z modyfikatorem **static** są atrybutem klasy, a nie obiektu. Oznacza to, że można się dobrać do takiego pola bez potrzeby kreacji obiektu i odwoływania się za jego pomocą do pola.

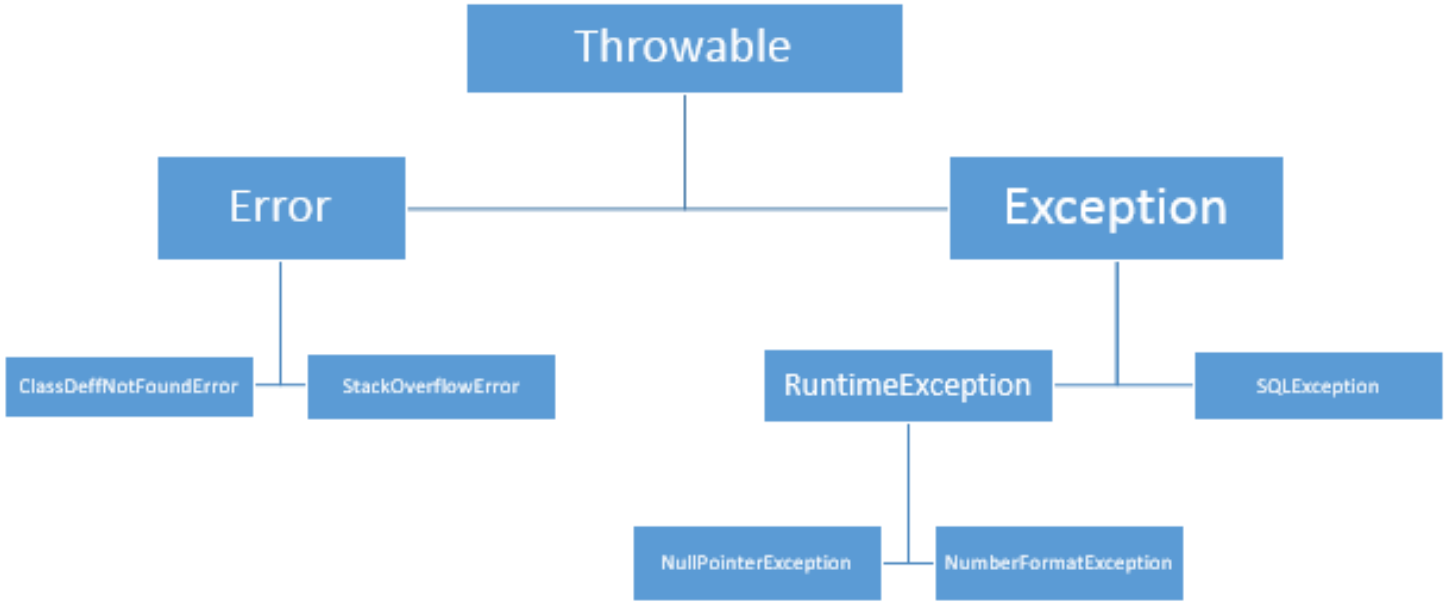
.

- **Metody statyczne**
- Metody statyczne nie działają na obiektach i za ich pomocą nie można operować na składowych obiektów. Mają dostęp do pól statycznych swoich klas.

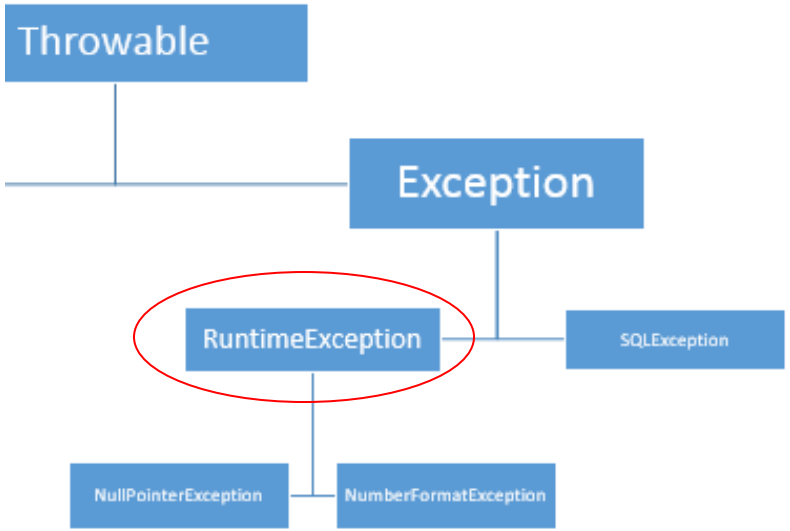
```
public class Calculator {
    public static int version = 6;

    public static final String name = "KALKULATOR";

    public static int add(int a, int b){
        return a+b;
    }
}
```

Wyjątki

- Wyjątek (ang. exception) jest specjalną klasą.
- Jest ona specyficzna ponieważ w swoim łańcuchu dziedziczenia ma klasę java.lang.Throwable.
- Instancje, które w swojej hierarchii dziedziczenia mają tę klasę mogą zostać „rzucone” (ang. throw) przerywając standardowe wykonanie programu.



RuntimeException



- Wyjątki klasy RuntimeException nie muszą być obsługiwane.
- Każdy z pozostałych wyjątków musi zostać obsłużony przez programistę.

```
public static int divide(int a, int b){
    if (b==0){
        throw new RuntimeException("Nie dziel przez zero");
    }
    return a/b;
}
```

Obsługa wyjątków

Klauzula throws

```
public static int divide(int a, int b) throws Exception{
    if (b==0){
        throw new Exception("Nie dziel przez zero");
    }
    return a/b;
}
```

Obsługa wyjątku poprzez blok try - catch

```
public static int divide(int a, int b){
    try {
        if (b == 0) {
            throw new Exception("Nie dziel przez zero");
        }
        return a / b;
    } catch (Exception e){
        return 0;
    }
}
```

Obsługa wyjątków – blok finally

```
public static int divide(int a, int b) {
    int c = 0;
    try {
        if (b == 0) {
            throw new Exception("Nie dziel przez zero");
        }
        c = a / b;
    } catch (Exception e) {
        System.out.println(e.getMessage());
    } finally {
        System.out.println("FINALLY");
    }
    return c;
}
```

Blok finally możemy umieścić po try. Kod wewnątrz tego bloku zawsze zostanie wykonany.
W rzeczywistości blok try nie musi mieć żadnej klauzuli catch jeśli ma blok finally.

Najpopularniejsze wyjątki

- NullPointerException — rzucany kiedy próbujesz wywołać metodę na zmiennej, której wartość to null
- IllegalArgumentException — rzucany, kiedy przekazywany argument jest z jakiegoś powodu nieprawidłowy (walidacja wewnątrz metod)
- IOException (wyjątki po nim dziedziczące) — rzucany w przypadku problemów z systemem wejścia/wyjścia, czyli najogólniej rzecz ujmując, kiedy wystąpi problem przy pracy z plikami lub z transmisją danych za pośrednictwem Internetu
- NumberFormatException — rzucany, kiedy próbujemy zamienić na liczbę np. obiekt typu String, który zawiera nie tylko cyfry
- IndexOutOfBoundException — rzucany, kiedy próbujemy się odwołać do nieistniejącego elementu tablicy lub listy

Propagacja wyjątków

- Wyjątek jest zawsze łapany przez pierwszy możliwy blok catch – taki który obsługuje wyjątki wskazanej klasy lub nadrzędne.

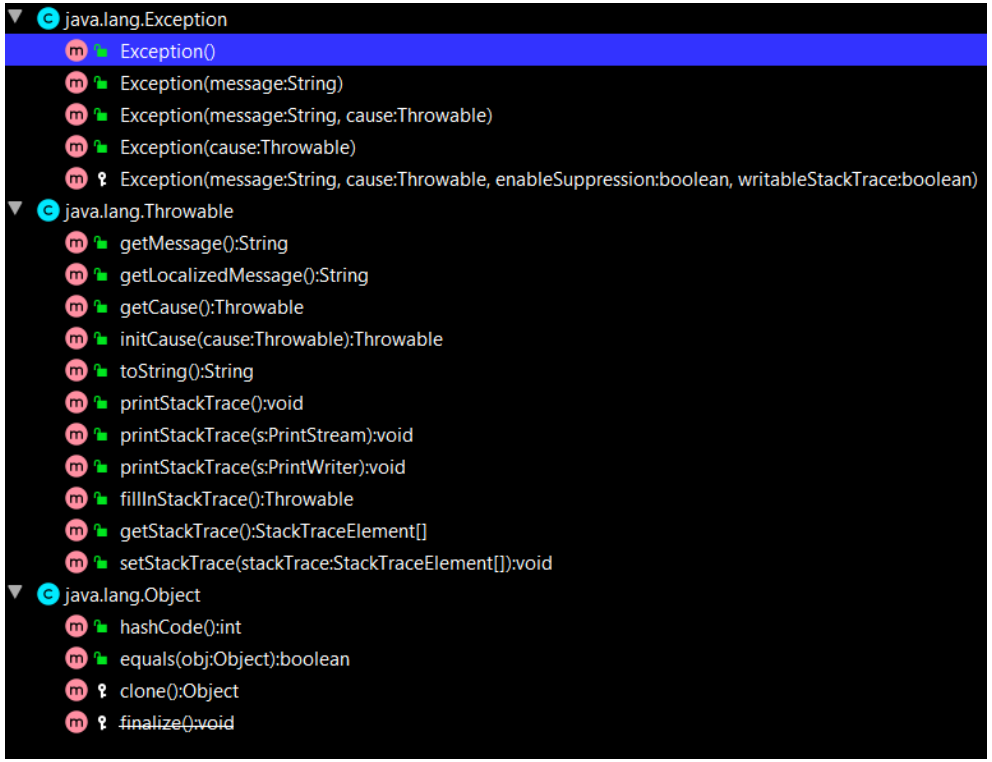
```java
public static int divide(int a, int b) {
    int c = 0;
    try {
        throw new .........;
    } catch (ArithmeticException e) {
        System.out.println("ArithmeticException "+e.getMessage());
    } catch (RuntimeException e) {
        System.out.println("RuntimeException "+e.getMessage());
    } catch (Exception e) {
        System.out.println("Exception "+e.getMessage());
    }
    return c;
}
```

```java
try {
    try {
        // Tu mamy wyjątek (ParseException) - przerwanie bloku try
        System.out.println("Chyba wszystko OK"); //instrukcja się nie wykona
    }
    catch (IOException e) {  // Nie pasuje
        e.printStackTrace();
    }
    catch (NumberFormatException e) {  // Nie pasuje - przerywa blok try!!!
        e.printStackTrace();
    }
}
catch (NullPointerException e) {  // Nie pasuje
    e.printStackTrace();
}
catch (RuntimeException e) { // Nie pasuje
    e.printStackTrace();
}
catch (Exception e) {  // Pasuje - wykonanie bloku catch
    e.printStackTrace();
}
```

Deklarowanie własnych wyjątków

- Aby zadeklarować własny wyjątek rozszerzamy klasę Exception lub inną klasę wyjątku oraz przesłaniamy odpowiednie metody.

```
public class DivideException extends Exception {
    @Override
    public String getMessage() {
        return "Nie dziel przez 0";
    }
}
```



ZADANIA -Wypisać stany stosów i narysować stertę

```
public class Point {
    private double x, y;
    public Point(double nx, double ny) {
        x = nx;
        y = ny;

    }

    public void move(double dx, double dy) {
        ; //mp1
        x = x + dx;
        y = y + dy;
        ;//mp2
    }

}
```

```
public class Line {
        private Point p1, p2;

        public void move(double dx, double dy) {
                p1.move(dx, dy);
                p2.move(dx, dy);
        }
        public Line(){
                p1 = new Point(0,0);
                p2 = new Point(0,0);
        }
        public Line(double x1, double y1, double x2, double  y2) {
                p1 = new Point(x1, y1);
                p2 = new Point(x2, y2);
        }
        public static void main(String[] args) {
                double  x1 = 1, y1 = 2, x2 = 3, y2 = 5;     //l0
                Line l = new Line(x1, y1, x2, y2);          //ml1
                l.move(2, 1);                               //ml2
        }
}
```

ZADANIA

1. Zaimplementuj klasę Factorial (silnia) z metodą factorial(int x) liczącą silnię x. Jaką sygnaturę powinna mieć ta metoda.
   a) Zachowując tę sygnaturę, zmodyfikuj tę implementację tak, aby dla x < 0 metoda nie zwracała wartości
   b) Zaimplementuj metodę factorial1, która dla liczb ujemnych będzie rzucać wyjątek MyException. MyException ma bezpośrednio rozszerzać Exception. Jak trzeba zmienić sygnaturę metody factorial?
   c) Jaka jest różnica pomiędzy tym a poprzednim rodzajem wyjątków?
   d) Przetestuj obie metody implementując metodę main tak, żeby w niej metody były sprawdzane dla ciągu argumentów typu: -3, -1, 0, 1, 2, 4, 5. Wskazówka: można użyć metody toString.
   e) Użyj metody printStackTrace do pokazania zawartości stosu wywołań w momencie rzucania wyjątku.

ZADANIA

2. Zaimplementuj klasę ThreeAttempts, która sczytuje z konsoli liczbę float i wypisuje ją w postaci np. „x = 1.2” (zobacz slajdy z pierwszego wykładu).

   Metoda ta powinna dopuszczać maksymalnie dwa błędy w typie danych, np. jeśli podczas pierwszej próby wpisany będzie string „abc”, to jest to błąd i metoda powinna zażądać innej danej i tak maksymalnie dwa razy. Metoda ma kończyć swoje wykonanie po pierwszym wprowadzeniu float lub po trzech nieudanych próbach.

ZADANIA

3. Zaimplementuj klasy Point, Line, Polygon, Group. Przedostatnia klasa ma implementować wielokąty wyznaczone przez pewną liczbę punktów. Ostatnia klasa ma reprezentować grupy figur (jak w przypadku edytorów typu PowerPoint). Wszystkie klasy mają implementować interfejs Figur (lub klasę abstrakcyjną z abstrakcyjnymi metodami min. equals(Object o) żeby wymusić nadpisanie). Zaimplementuj również klasę Group grupująca figury i również implementującą Figure. Figure ma posiadać metody:
   - move(double dx, double dy) przesuwającą daną figurę
   - flip() przerzucająca dana figurę
   - equals(Object o) sprawdzającą, czy figury są równe
   - toString() zwracająca łańcuch znaków (String) reprezentujący daną figurę.
4. W klasie Polygon zaimplementuj metody do wyznaczania obwodu i pola. W sytuacji gdy to niemożliwe rzuć utworzony przez siebie odpowiedni wyjątek.