Neues Tut. Do 10-12 im MA 645

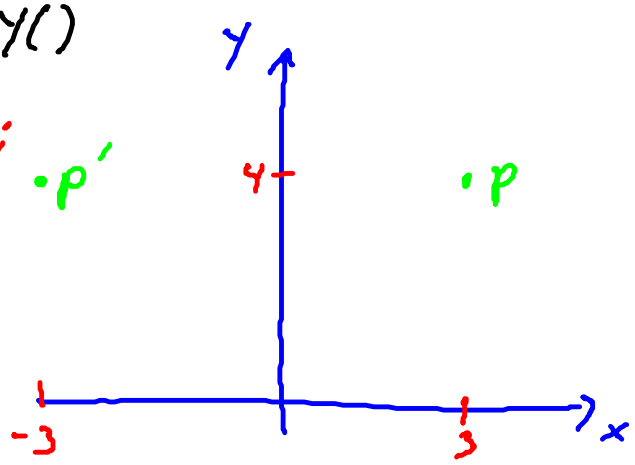
Letzte VL: Klasse Point repräsentiert Punkte in der Ebene

Methoden in Point: Konstruktoren, Set- und Get-Methoden, toString(), mirrorY()

Korrektur: `Point p = new Point(3,4);`

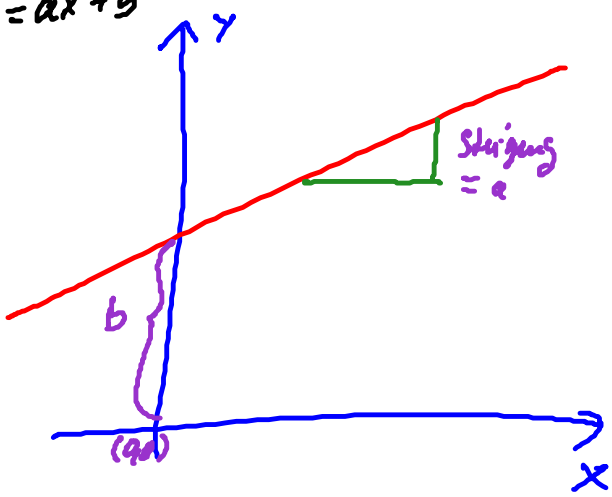
~~`Point p' = p.mirrorY();`~~ falsch

`Point p' = p.clone();`
`p'.mirrorY();`



Zusätzliche Klasse Line

↑ Objekte sind Geraden $y = ax + b$ in der Ebene

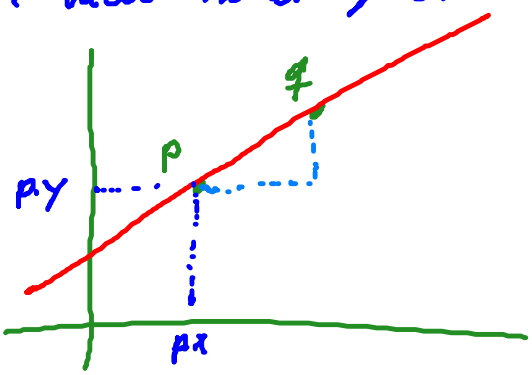



```
public class Line {

        // a line is a function y = ax + b
        // we represent it with the object variables
        // slope for a and offset for b

        // object variables

        private double slope; // slope
        private double offset; // offset at x = 0


    public Line() { // default constructor
            this.slope = 0;
            this.offset = 0;
    }

    public Line(double slope, double offset) { // standard constructor
            this.slope = slope;
            this.offset = offset;
    }
```

Willkürlich wird die Gerade gewählt, die auf der x-Achse liegt.

Wollen jetzt weiteren Konstruktor implementieren, der zwei

Punkte p, q übergeben bekommt und die Gerade erzeugt, die durch diese Punkt geht.



Dazu muss aus p und q die Steigung und der Offset der Geraden berechnet werden.

p.x x-Koordinate von p
p.y y-Koordinate von p

analog für q

$$slope := (q.y - p.y) / (q.x - p.x)$$

Berechnung des offsets: Es gilt

$$p.y = offset + p.x \cdot slope$$

$$\Rightarrow offset = p.y - p.x \cdot slope$$

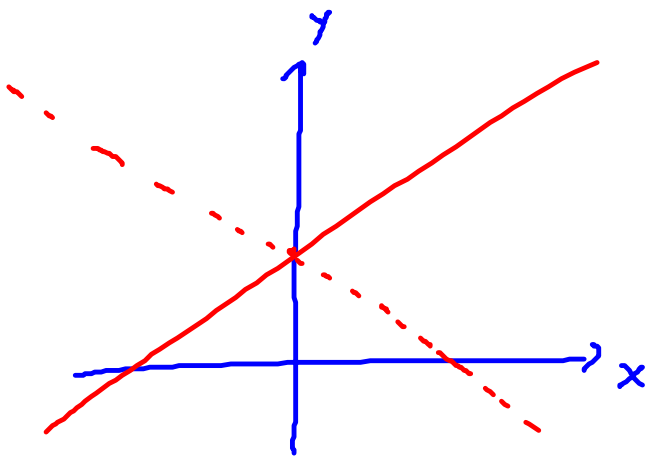
```
// constructor when two points are given
public Line(Point p, Point q) {
        this.slope = (p.getY() - q.getY()) / (p.getX() - q.getX());
        this.offset = p.getY() - this.slope * p.getX();
}
```

Klasse Point wird hier verwendet

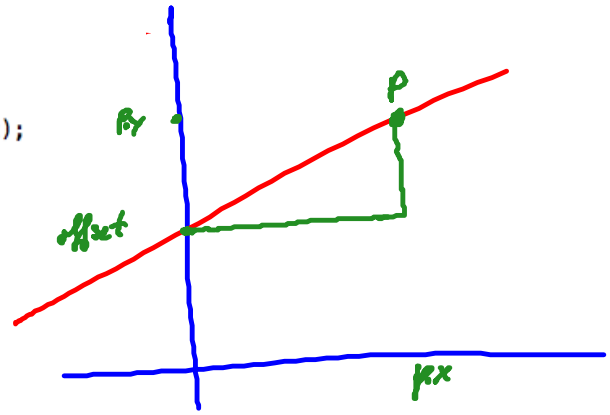
↖ get-Methode der Klasse Point.

```
// set methods

public void setSlope(double slope) {
        this.slope = slope;
}

public void setOffset(double offset) {
        this.offset = offset;
}

// get methods

public double getSlope() {
        return this.slope;
}

public double getOffset() {
        return this.offset;
}


 // mirror line at y-axis

 public void mirrorY() {
         this.slope = -this.slope;
 }


 // string representation

 public String toString() {
```

wie bei Klasse Point



```
        return "y = " + this.slope + "*x + " + this.offset;
    }


    // evaluation at x
    public double evaluate(double x) {
        return this.slope*x + this.offset;
    }



    // test if this line contains point p
    public boolean contains(Point p) {
        return (this.slope * p.getX()) == (p.getY() - this.offset);
    }
}
```

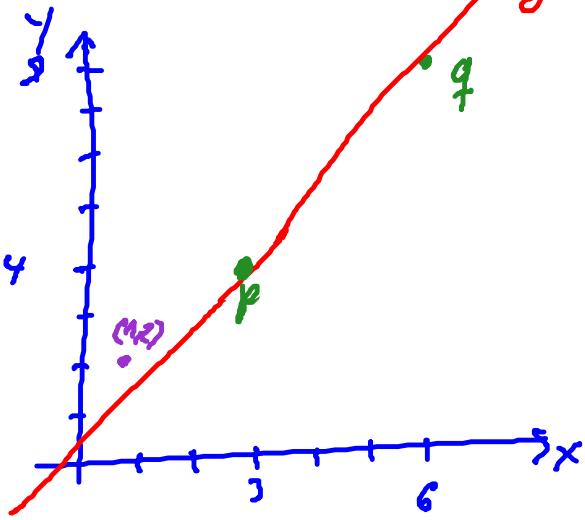


Programm/Klasse Line auf www.

Testen mit Programm/Klasse TestLine (im selben Directory):

```
public class TestLine {

        public static void main(String[] args) {


            Point p = new Point(3,4);
            Point q = new Point(6,8);

            Line g = new Line(p,q);

            System.out.println( g.getSlope() );
            System.out.println( g.getOffset() );
            System.out.println( g.toString() );

            System.out.println( g.contains(p) );
            System.out.println( g.contains(new Point(1,2)));

        }


}
```

→ 1,333...
→ 0
y = 1.33... * x + 0
→ true
→ false




```
Terminal — bash — 64×18

dude-5:Point peis$ javac TestLine.java
dude-5:Point peis$ java TestLine
1.3333333333333333
0.0
y = 1.3333333333333333*x + 0.0
true
false
dude-5:Point peis$
```

Es gibt jedoch Probleme mit dem Konstruktor, der eine Gerade aus zwei gegebenen Punkten konstruiert:

Wenn p und q dieselbe x-Koordinate haben, wird durch 0 dividiert. (Gerade hat unendliche/undefinierte Steigung).

Möchten dies abfangen!

Jetzt richtig mit Java Exceptionhandling:

Dazu können Methoden Exceptions werfen.

Syntax:

```
Rückgabetyp methodenName(Parameterliste)
        throws Exceptionklasse { ...
        .
        .
        .
        .
    if ( Bed. für Exception is true )
        throw new Exceptionklasse(String);
        .
        .
}
```

- Exceptionklasse: hier eine spezielle Exceptionklasse einfügen, evtl. mehrere durch Kommata getrennt
- throw: bricht Methode ab (wie return)
- new Exceptionklasse: erzeuge neues Objekt der Exceptionklasse
- String: Fehlermeldung

```java
// constructor when two points are given
public Line(Point p, Point q) throws IllegalArgumentException {

    if (p.getX() == q.getX()) throw new
        IllegalArgumentException("points have same x-coordinate,"
                                 +"\nslope is undefined.");

        this.slope = (p.getY() - q.getY()) / (p.getX() - q.getX());
        this.offset = p.getY() - this.slope * p.getX();
}
```

neu

Abfrage der Exceptions ist auf Benutzerebene mit "try and catch"

```java
try {
    statements
}
catch (Exceptionklasse_1 Exceptionvariable_1) {
    Statements für Exceptionbehandlung
}

...

catch (Exceptionklasse_k Exceptionvariable_k) {
    Statements für Exceptionbehandlung
}
// Falls nötig kann ein finally Block angehängt werden
finally {
    weitere Anweisungen für Exceptionbehandlung
}
```

allg. Syntax für try + catch

```java
public class TestLine2 {

        public static void main(String[] args) {

            try {

                Point p = new Point(3,4);
                Point q = new Point(3,8);

                Line g = new Line(p,q);

                System.out.println( g.toString() );

            }
            catch (IllegalArgumentException e) {
                System.out.println( e.getMessage());
            }

        }
}
```

p und q haben gleiche x-Koordinate

Hier wird exception geworfen.

Danach wird direkt hierhin gesprungen.

```
Terminal — bash — 62×13

dude-5:Point peis$ javac TestLine2.java
dude-5:Point peis$ java TestLine2
points have same x-coordinate,
slope is undefined.
dude-5:Point peis$
```