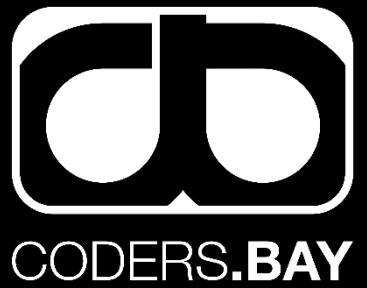
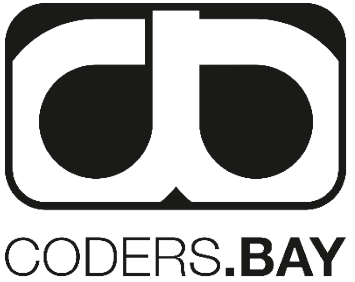


ENCAPSULATION

Objekt Orientierung - Datenkapselung

Access Modifier



- **private**: nur innerhalb der Klasse verfügbar

```
private String address;
```

- **package-private**: nur innerhalb von Klassen im gleichen Package verfügbar

```
String phoneNumber;
```

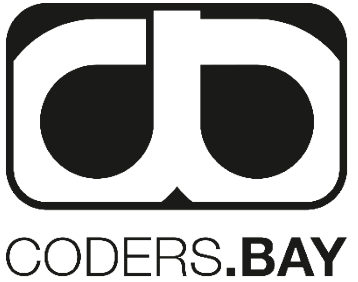
- **protected**: nur von Klassen in der Hierarchie verfügbar (mehr dazu im nächsten Kapitel - **Vererbung**)

```
protected String birthday;
```

- **public**: öffentlich verfügbar

```
public String name;
```

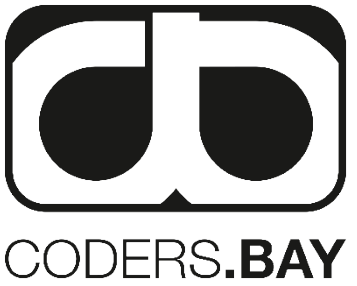
Getter & Setter



Beim Prinzip der Getter & Setter, wird der Zugriff auf das Attribut der Klasse selbst begrenzt (durch das private oder protected keyword) und dafür werden Methoden zum Lesen und Schreiben dieses Wertes verwendet.

```
public class Person {
   public String name;
}
```

Getter & Setter

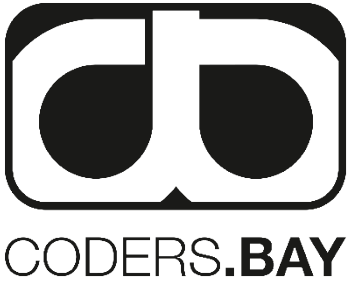



```
public class Person {
   private String name;
   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
}
```

BUT WHY?

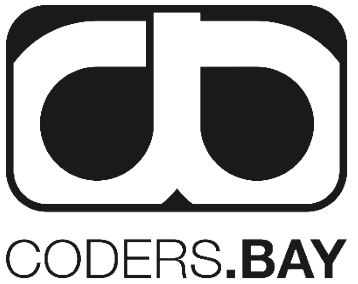
Im IntelliJ kannst du dir Getter & Setter durch Drücken von **Alt+Einfg** generieren lassen

Getter & Setter - Wozu?



- Kapselung von Verhalten welches eng mit dem Zugriff auf eine Variable verknüpft ist - z.B. Validierung
- um die interne Struktur zu verbergen und die Repräsentation nach “außen” bei zu behalten
- Unterschiedliche Zugriffslevel für das Lesen und Schreiben einer Variable (get public, set protected)
- für Zusammenarbeit mit manchen Bibliotheken - Mocking, Serialisierung, ...

Exercise Time!

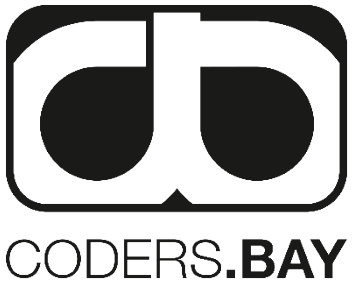


Distanzberechnung in einem kartesischen Koordinatensystem!

Wir schreiben eine Klasse **Point**, die einen Punkt im kartesischen Koordinatensystem darstellt. Also einen Punkt mit einer X- und Y-Koordinate.

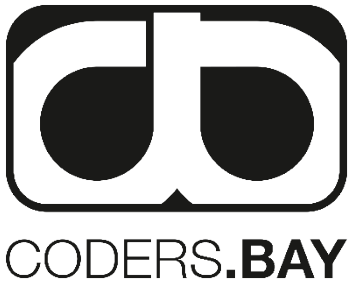
In der Klasse in der sich auch unsere main-Methode befindet schreiben wir eine Methode, welche die Distanz zwischen zwei Punkten berechnen kann.

Point Klasse



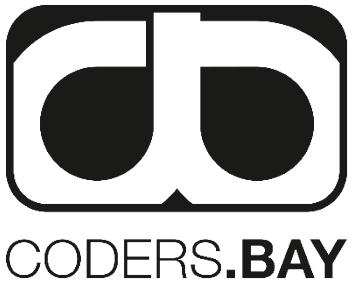

```
public class Point {
   private Integer x;
   private Integer y;

   public Integer getX() {
      return x;
   }

   public void setX(Integer x) {
      this.x = x;
   }

   public Integer getY() {
      return y;
   }

   public void setY(Integer y) {
      this.y = y;
   }
}
```

Distanzberechnung




```
public static Double calculateDistance(Point pointA,
                                       Point pointB) {
   return Math.sqrt(
           Math.pow(pointA.getX() - pointB.getX(), 2.0) +
           Math.pow(pointA.getY() - pointB.getY(), 2.0)
   );
}
```

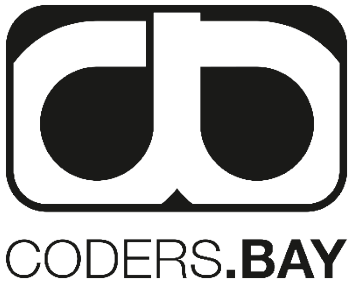
Exercise Time!



Wir wollen jetzt unser Koordinatensystem begrenzen - wir erlauben es nur im ersten Quadranten des Koordinatensystems Punkte anzulegen - also dort wo x und y größer 0 ist.

Dank der Setter können wir hier einfach eine Validierung einbauen!

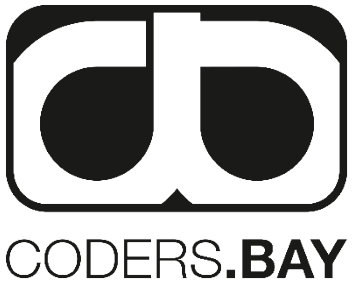
Exercise Time!



Als nächstes wollen wir die interne Repräsentation unserer Variablen ändern - wir möchten uns Polarkoordinaten speichern und keine kartesischen Koordinaten mehr.

Und das möglichst ohne die **calculateDistance** Methoden ändern zu müssen!

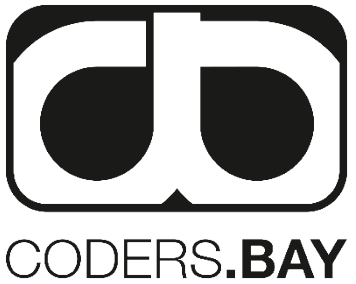
Das gleiche oder das selbe?




```
Point pointA = new Point(2, 3);
Point pointB = new Point(2, 3);
System.out.printf("pointA == pointB %s\n", pointA == pointB);
System.out.printf("pointA.equals(pointB) %s\n", po
Point pointC = new Point(1, 3);
System.out.printf("pointC == pointB %s\n", pointC
System.out.printf("pointC.equals(pointB) %s\n", po
```

```
pointA == pointB false
pointA.equals(pointB) false
pointC == pointB false
pointC.equals(pointB) false
```

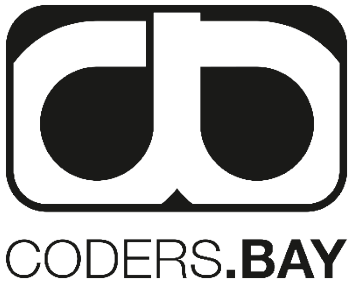
Equals




```java
@Override
public boolean equals(Object o) {
   Point other = (Point) o;
   return Objects.equals(other.x, this.x) &&
          Objects.equals(other.y, this.y);
}
```

```
pointA == pointB false
pointA.equals(pointB) false
pointC == pointB false
pointC.equals(pointB) false
```

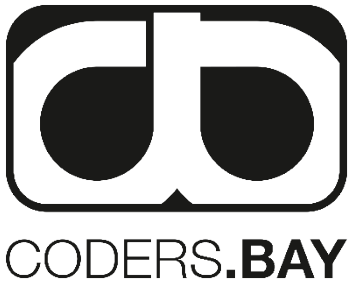
Equals




```
System.out.println("1.000 == 1.000 is " +
  (new Integer(1000) == new Integer(1000)));
System.out.println("1.000 equals 1.000 is " +
  (new Integer(1000).equals(new Integer(1000))));
System.out.println("1.000 == 1.000 is " +
  (1000 == 1000));
```

```
1.000 == 1.000 is false
1.000 equals 1.000 is true
1.000 == 1.000 is true
```

Equals und ==

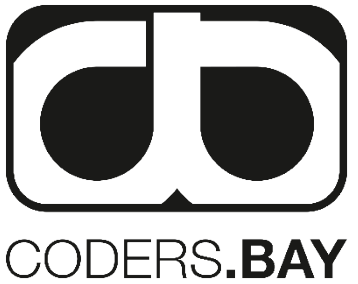


Mit Equals kann man definieren, wann zwei Objekte als gleich bezeichnet werden, auch wenn eventuell das Objekt dahinter nicht genau das gleiche ist.

Der Operator == überprüft immer ob es sich wirklich um das gleiche Objekt handelt und verwendet die equals Logik **nicht**!

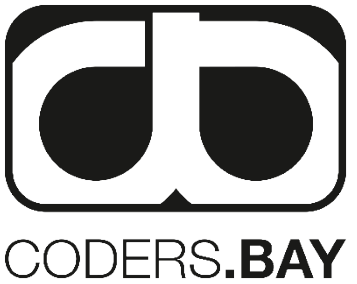
Im IntelliJ kannst du dir Getter & Setter durch Drücken von **Alt+Einfg** generieren lassen

Hashcode



Jede Klasse hat eine Hashcode Methode, die einen Integer zur repräsentation des Objekts zurückgibt.
Hashcodes sind deterministisch, d.h. die hashcode Methode muss für jeden Aufruf auf einem Objekt immer das gleiche Ergebnis liefern, der hashcode muss allerdings nicht immer eindeutig sein - 2 unterschiedliche Objekte können den selben hashcode liefern.
Verwendet wird der Hashcode zum Beispiel für Zugriffe in einem Hashset.

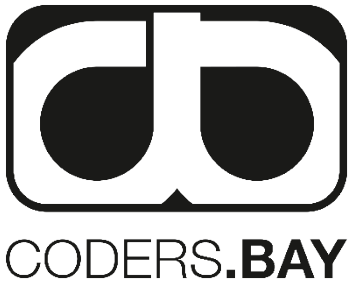
Schreibs auf die Konsole




```
Point pointA = new Point(2, 3);
System.out.println("Point A: " + pointA);
Point pointB = new Point(2, 3);
System.out.println("Point B: " + pointB);
Point pointC = new Point(1, 3);
System.out.println("Point C: " + pointC);
```

```
Point A: Point@3498ed
Point B: Point@1a407d53
Point C: Point@3d8c7aca
```

toString

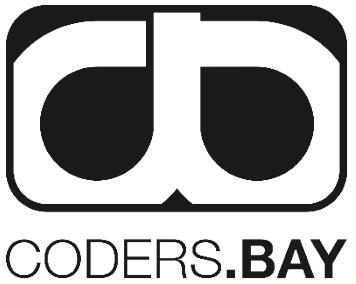


Die Default-Implementierung in der **Object** Klasse:

```
public String toString() {
   return getClass().getName() + "@" +
         Integer.toHexString(hashCode());
}
```

```
Point A: Point@3498ed
Point B: Point@1a407d53
Point C: Point@3d8c7aca
```

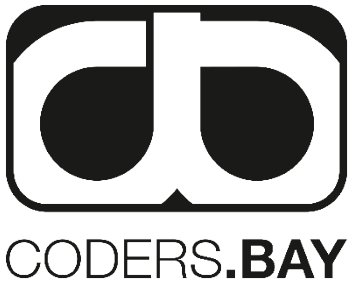
toString




```java
@Override
public String toString() {
   return "Point{" +
          "x=" + x +
          ", y=" + y +
          "}";
}
```

```
Point A: Point{x=2, y=3}
Point B: Point{x=2, y=3}
Point C: Point{x=1, y=3}
```

Zusammenfassung



- Access Modifier regeln den Zugriff und können auf Variablen und Methoden angewendet werden.
- Getter und Setter dienen der Kapselung der internen Struktur von Variablen
- Equals und Hashcode zum Definieren der Gleichheitslogik
  - objektA.equals(objektB) *vs* objektA == objektB
- toString Methode für Logging sehr wertvoll