Lösungsvorschlag: Konstruktoren und Objekte

TODO 1 & 2

```
        // TODO 1:  Erstellen Sie einen eigenen Datentyp "Boot" jedes Boot
hat folgende Eigenschaften:
        //          Marke, Farbe, Kaufpreis, Seriennummer und ist im Wasser
oder nicht
        //          Denken Sie daran, die Attribute als private zu
markieren!

        // TODO 2:  Erstellen Sie für das Boot zwei Konstruktoren
        //          Boot(<Datentyp> marke, <Datentyp> seriennummer)
        //          Boot(<Datentyp> marke, <Datentyp> farbe, <Datentyp>
kaufpreis,
        //                          <Datentyp> seriennummer, <Datentyp>
imWasser)


/**
 * Datentyp Boot
 *
 * @author Kevin Maurizi
 * @since 2042.01.01
 * @version 0.1
 */
public class Boot {
    private String marke;
    private String farbe;
    private double kaufpreis;
    private String seriennummer;
    private boolean imWasser;

    /**
     * Konstruktor mit den unveränderbaren Werten
     * @param marke
     * @param seriennummer
     */
    public Boot(String marke, String seriennummer){
        this.marke = marke;
        this.seriennummer = seriennummer;
    }

    /**
     * Kompletter Konstruktor
```

```
     * @param marke
     * @param farbe
     * @param kaufpreis
     * @param seriennummer
     * @param imWasser
     */
    public Boot(String marke, String farbe, double kaufpreis, String
seriennummer, boolean imWasser) {
        this.marke = marke;
        this.farbe = farbe;
        this.kaufpreis = kaufpreis;
        this.seriennummer = seriennummer;
        this.imWasser = imWasser;
    }
}
```

TODO 4

```
        // TODO 4:  Erstellen Sie getter und setter für die Attribute
        //          Überlegen Sie sich, ob es Sinn macht, für jedes Attribut
einen Setter zu machen.

/**
 * Datentyp Boot
 *
 * @author Kevin Maurizi
 * @since 2042.01.01
 * @version 0.1
 */
public class Boot {
    private String marke;
    private String farbe;
    private double kaufpreis;
    private String seriennummer;
    private boolean imWasser;

    /**
     * Konstruktor mit den unveränderbaren Werten
     * @param marke
     * @param seriennummer
     */
    public Boot(String marke, String seriennummer){
        this.marke = marke;
        this.seriennummer = seriennummer;
    }

    /**
```

```
     * Kompletter Konstruktor
     * @param marke
     * @param farbe
     * @param kaufpreis
     * @param seriennummer
     * @param imWasser
     */
    public Boot(String marke, String farbe, double kaufpreis, String
seriennummer, boolean imWasser) {
        this.marke = marke;
        this.farbe = farbe;
        this.kaufpreis = kaufpreis;
        this.seriennummer = seriennummer;
        this.imWasser = imWasser;
    }

    public void setFarbe(String farbe) {
        this.farbe = farbe;
    }

    public String getFarbe() {
        return farbe;
    }

    public void setImWasser(boolean imWasser) {
        this.imWasser = imWasser;
    }

    public boolean isImWasser() {
        return imWasser;
    }

    public void setKaufpreis(double kaufpreis) {
        this.kaufpreis = kaufpreis;
    }

    public double getKaufpreis() {
        return kaufpreis;
    }

    // Für Marke und Seriennummer keinen Setter, da diese nicht geändert
werden können.
    public String getMarke() {
        return marke;
    }

    public String getSeriennummer() {
        return seriennummer;
    }
}
```

TODO 6

```
       // TODO 6:  Erstellen Sie die Methode printInfos() im Boot. Diese Methode soll alle Infos über das Boot ausgeben.


/**
 * Datentyp Boot
 *
 * @author Kevin Maurizi
 * @since 2042.01.01
 * @version 0.1
 */
public class Boot {
    private String marke;
    private String farbe;
    private double kaufpreis;
    private String seriennummer;
    private boolean imWasser;

    /**
     * Konstruktor mit den unveränderbaren Werten
     * @param marke
     * @param seriennummer
     */
    public Boot(String marke, String seriennummer){
        this.marke = marke;
        this.seriennummer = seriennummer;
    }

    /**
     * Kompletter Konstruktor
     * @param marke
     * @param farbe
     * @param kaufpreis
     * @param seriennummer
     * @param imWasser
     */
    public Boot(String marke, String farbe, double kaufpreis, String seriennummer, boolean imWasser) {
        this.marke = marke;
        this.farbe = farbe;
        this.kaufpreis = kaufpreis;
        this.seriennummer = seriennummer;
        this.imWasser = imWasser;
    }
```

```
    public void setFarbe(String farbe) {
        this.farbe = farbe;
    }

    public String getFarbe() {
        return farbe;
    }

    public void setImWasser(boolean imWasser) {
        this.imWasser = imWasser;
    }

    public boolean isImWasser() {
        return imWasser;
    }

    public void setKaufpreis(double kaufpreis) {
        this.kaufpreis = kaufpreis;
    }

    public double getKaufpreis() {
        return kaufpreis;
    }

    // Für Marke und Seriennummer keinen Setter, da diese nicht geändert
werden können.
    public String getMarke() {
        return marke;
    }

    public String getSeriennummer() {
        return seriennummer;
    }

     /**
     * Druckt alle Infos zum Boot aus
     */
    public void printInfos(){
        System.out.println("Marke: " + marke);
        System.out.println("Seriennummer: " + seriennummer);
        System.out.println("Farbe: " + farbe);
        System.out.println("Preis: " + kaufpreis);
        System.out.println("ImWasser: " + (imWasser?"Ja":"Nein"));
    }
}
```

Bedingungsoperator

```
imWasser?"Ja":"Nein"
```

Der ternäre Operator kann eine if-else-Verzweigung ersetzen und weist meist einer Variablen einen Wert in Abhängigkeit vom Ergebnis einer Bedingungsprüfung zu. Er ist der einzige Operator, der mit drei Operanden arbeitet, wird oft auch Bedingungsoperator genannt und besitzt folgenden allgemeinen Aufbau:

```
bedingung ? wert1 : wert2
```

Die Variablenzuweisung erfolgt dann in der Form

```
variable = bedingung ? wert1 : wert2
```

`bedingung` muss immer ein boolscher Ausdruck sein. Er entscheidet über die Wertzuweisung. Ist er true, so wird der Wert nach dem Fragezeichen zugewiesen, ansonsten der Wert nach dem Doppelpunkt.

In diesem Fall wird der Wert nicht einer Variablen zugewieden, sondern direkt im `System.out.println` ausgegebn.

```
System.out.println("ImWasser: " +
(imWasser?"Ja":"Nein"));
```

Vor und Nachteile: Er kann Anweisungen sehr kurz fassen, allerdings - wie in solchen Fällen häufig - auf Kosten der Lesbarkeit, besonders wenn die Anweisungen komplexer werden.

TODO 8

```
        // TODO 8:  Erstellen Sie 3 Boote und fügen Sie diese einer
Bootsliste hinzu

        Boot boot = new Boot("Bavaria","Blau",154000,"BAV25454ASD",true);
        Boot boot1 = new Boot("Hallberg
Rassy","Weiss",256000,"RASS458778",false);

        ArrayList<Boot> boote = new ArrayList<>();
        boote.add(boot);
        boote.add(boot1);
        boote.add(new Boot("Lagoon","Hellblau",348000,"LAG122225",true));
```

Konstruktor ohne Variable

```
boote.add(new
```

```
Boot("Lagoon","Hellblau",348000,"LAG122225",true));
```

Die Erstellung eines Bootes mit dem Konstruktor kann auch direkt in der `add()` Methode der Liste erfolgen.

TODO 9

```
        // TODO 9:  Iterieren Sie (Schleife) über diese Liste und geben Sie die Infos über alle Boote aus.
        //          Verwenden Sie dazu die Methode printInfos()

        // Variante 1
        for (Boot b : boote) {
            b.printInfos();
        }

        // Variante 2
        for (int i = 0; i < boote.size(); i++) {
            boote.get(i).printInfos();
        }
```