

# ENUMS

# Aufgabe:

Folgende Klasse ist gegeben:

```
import java.util.List;

public class Pizza {

    private Pizzagroesse pizzagroesse;
    private List<Zutat> zutaten;

    public Pizza(Pizzagroesse pizzagroesse, List<Zutat> zutaten){
        this.pizzagroesse = pizzagroesse;
        this.zutaten = zutaten;
    }
}
```

Wie würdet ihr die Klasse Pizzagroesse implementieren?

# Erster Gedanke für die Implementierung

```
public class Pizzagroesse {  
  
    private String groesse;  
    public Pizzagroesse(String groesse){  
        this.groesse = groesse;  
    }  
  
    public String getGroesse(){  
        return this.groesse;  
    }  
}
```

# Erster Gedanke für die Implementierung

```
import java.util.List;

public class Pizza {

    private Pizzagroesse pizzagroesse;
    private List<Zutat> zutaten;

    public Pizza(Pizzagroesse pizzagroesse, List<Zutat> zutaten){
        this.pizzagroesse = pizzagroesse;
        this.zutaten = zutaten;
    }
}
```

```
public class Pizzagroesse {

    private String groesse;
    public Pizzagroesse(String groesse){
        this.groesse = groesse;
    }

    public String getGroesse(){
        return this.groesse;
    }
}
```

Wie würdet ihr eine Pizza erstellen?

# Erster Gedanke für die Implementierung

```
import java.util.List;

public class Pizza {

    private Pizzagroesse pizzagroesse;
    private List<Zutat> zutaten;

    public Pizza(Pizzagroesse pizzagroesse, List<Zutat> zutaten){
        this.pizzagroesse = pizzagroesse;
        this.zutaten = zutaten;
    }
}
```

```
public class Pizzagroesse {

    private String groesse;
    public Pizzagroesse(String groesse){
        this.groesse = groesse;
    }

    public String getGroesse(){
        return this.groesse;
    }
}
```

```
Pizzagroesse groesse = new Pizzagroesse(„Klein“);
Pizza p = new Pizza(groesse, zutatenliste);
```

# Erster Gedanke für die Implementierung

```
import java.util.List;

public class Pizza {

    private Pizzagroesse pizzagroesse;
    private List<Zutat> zutaten;

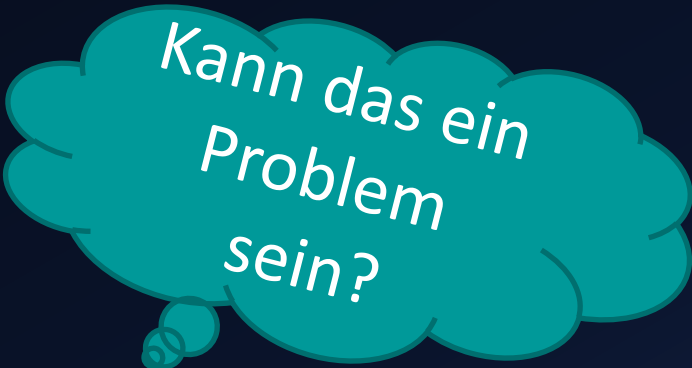
    public Pizza(Pizzagroesse pizzagroesse, List<Zutat> zutaten){
        this.pizzagroesse = pizzagroesse;
        this.zutaten = zutaten;
    }
}
```

```
public class Pizzagroesse {

    private String groesse;
    public Pizzagroesse(String groesse){
        this.groesse = groesse;
    }

    public String getGroesse(){
        return this.groesse;
    }
}
```

```
Pizzagroesse groesse = new Pizzagroesse(„Klein“);
Pizza p = new Pizza(groesse, zutatenliste);
```



Kann das ein  
Problem  
sein?

# Erster Gedanke für die Implementierung

- Pro Pizza-Objekt muss ein Pizzagroesse-Objekt erstellt werden
- Es können nicht existierende Pizzagrößen erstellt werden (z.B. „Frosch“)



# Erster Gedanke für die Implementierung

- Pro Pizza-Objekt muss ein Pizzagroesse-Objekt erstellt werden
- Es können nicht existierende Pizzagrößen erstellt werden (z.B. „Frosch“)



- Wie kann man zumindest vermeiden, dass nicht-existierende Pizzagrößen **von außen** erstellt werden?

```
public class Pizzagroesse {  
    private String groesse;  
    public Pizzagroesse(String groesse){  
        this.groesse = groesse;  
    }  
  
    public String getGroesse(){  
        return this.groesse;  
    }  
}
```



# Die Zeit vor Enums – Problemlösung

```
public class Pizzagroesse {  
    private String groesse;  
  
    private Pizzagroesse(String groesse) {  
        this.groesse = groesse;  
    }  
}
```



Was ist  
anders?

# Die Zeit vor Enums – Problemlösung

```
public class Pizzagroesse {  
    private String groesse;  
  
    private Pizzagroesse(String groesse) {  
        this.groesse = groesse;  
    }  
}
```

Privater Konstruktor



Was ist  
anders?

# Die Zeit vor Enums – Problemlösung

```
public class Pizzagroesse {  
    private String groesse;  
  
    private Pizzagroesse(String groesse) {  
        this.groesse = groesse;  
    }  
}
```

Beim Versuch ein Pizzaobjekt zu erstellen:

```
Pizzagroesse groesse = new Pizzagroesse(„Klein“);  
Pizza p = new Pizza(groesse, zutaten);
```



Compilerfehler!

# Die Zeit vor Enums – Problemlösung

```
public class Pizzagroesse {  
    private String groesse;  
  
    private Pizzagroesse(String groesse) {  
        this.groesse = groesse;  
    }  
}
```

Beim Versuch ein Pizzaobjekt zu erstellen:

```
Pizzagroesse groesse = new Pizzagroesse(„Klein“);  
Pizza p = new Pizza(groesse, zutaten);
```



Compilerfehler!

Wie kann das trotzdem zugänglich machen?

# Die Zeit vor Enums – Problemlösung

```
public class Pizzagroesse {  
    private String groesse;  
  
    private Pizzagroesse(String groesse) {  
        this.groesse = groesse;  
    }  
}
```

Beim Versuch ein Pizzaobjekt zu erstellen:

```
Pizzagroesse groesse = new Pizzagroesse(„Klein“);  
Pizza p = new Pizza(groesse, zutaten);
```



Compilerfehler!

Wie kann das trotzdem zugänglich machen? -> Konstanten innerhalb der Klasse

# Die Zeit vor Enums – Problemlösung

```
public class Pizzagroesse {  
    private String groesse;
```

```
    private Pizzagroesse(String groesse) {  
        this.groesse = groesse;  
    }
```

```
    public static final Pizzagroesse KLEIN = new Pizzagroesse("Klein");  
    public static final Pizzagroesse MITTEL = new Pizzagroesse("Mittel");  
    ...  
}
```

} Objektkonstanten

# Die Zeit vor Enums – Problemlösung

```
public class Pizzagroesse {  
    private String groesse;  
  
    private Pizzagroesse(String groesse) {  
        this.groesse = groesse;  
    }  
  
    public static final Pizzagroesse KLEIN = new Pizzagroesse("Klein");  
    public static final Pizzagroesse MITTEL = new Pizzagroesse("Mittel");  
    ...  
}
```

```
import java.util.List;
```

```
public class Pizza {  
  
    private Pizzagroesse pizzagroesse;  
    private List<Zutat> zutaten;  
  
    public Pizza(Pizzagroesse pizzagroesse, List<Zutat> zutaten){  
        this.pizzagroesse = pizzagroesse;  
        this.zutaten = zutaten;  
    }  
}
```

Wie würden wir nun ein  
Pizza-Objekt erstellen?

# Die Zeit vor Enums – Problemlösung

```
public class Pizzagroesse {  
    private String groesse;  
  
    private Pizzagroesse(String groesse) {  
        this.groesse = groesse;  
    }  
  
    public static final Pizzagroesse KLEIN = new Pizzagroesse("Klein");  
    public static final Pizzagroesse MITTEL = new Pizzagroesse("Mittel");  
    ...  
}  
  
import java.util.List;  
  
public class Pizza {  
  
    private Pizzagroesse pizzagroesse;  
    private List<Zutat> zutaten;  
  
    public Pizza(Pizzagroesse pizzagroesse, List<Zutat> zutaten){  
        this.pizzagroesse = pizzagroesse;  
        this.zutaten = zutaten;  
    }  
}
```

Wie würden wir nun ein  
Pizza-Objekt erstellen?

```
Pizza p = new Pizza(Pizzagroesse.KLEIN, zutaten)
```



# Die Zeit vor Enums – Problemlösung

```
public class Pizzagroesse {  
    private String groesse;  
  
    private Pizzagroesse(String groesse) {  
        this.groesse = groesse;  
    }  
  
    public static final Pizzagroesse KLEIN = new Pizzagroesse("Klein");  
    public static final Pizzagroesse MITTEL = new Pizzagroesse("Mittel");  
    ...  
}  
  
import java.util.List;  
  
public class Pizza {  
  
    private Pizzagroesse pizzagroesse;  
    private List<Zutat> zutaten;  
  
    public Pizza(Pizzagroesse pizzagroesse, List<Zutat> zutaten){  
        this.pizzagroesse = pizzagroesse;  
        this.zutaten = zutaten;  
    }  
}
```

Wie würden wir nun ein  
Pizza-Objekt erstellen?

Pizza p = new Pizza(Pizzagroesse.KLEIN, zutaten)



Referenz auf die Konstante

# Die Zeit vor Enums – Problemlösung

```
public class Pizzagroesse {  
    private String groesse;  
  
    private Pizzagroesse(String groesse) {  
        this.groesse = groesse;  
    }  
  
    public static final Pizzagroesse KLEIN = new Pizzagroesse("Klein");  
    public static final Pizzagroesse MITTEL = new Pizzagroesse("Mittel");  
    ...  
}
```

## Zusammenfassung:

- Durch den privaten Konstruktor kann man von außerhalb keine eigenen Größen erstellen
- Es muss nicht pro Pizza ein Pizzagroessen – Objekt erstellt werden -> Zugriff auf das Objekt mittels Referenz

## Warum trotzdem nicht ideal?

- Code kann schnell sehr lang werden
- Klarheit und Lesbarkeit



# Die Zeit vor Enums – Problemlösung am Beispiel Schrauben und Nägel im Bauhaus

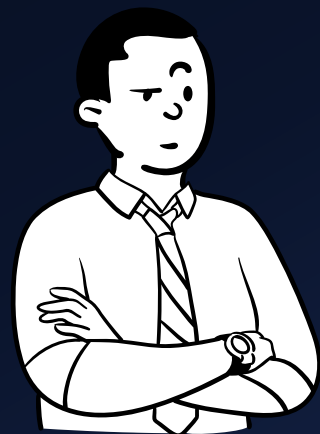
```
public class Schraube {
    private String name;
    private int laenge; // Länge in mm
    private double durchmesser; // Durchmesser in mm

    private Schraube(String name, int laenge, double durchmesser) {
        this.name = name;
        this.laenge = laenge;
        this.durchmesser = durchmesser;
    }

    public static final Schraube SCHRAUBE_1 = new Schraube("Schraube 1", 10, 1.5);
    public static final Schraube SCHRAUBE_2 = new Schraube("Schraube 2", 12, 2.0);
    public static final Schraube SCHRAUBE_3 = new Schraube("Schraube 3", 14, 2.5);
    public static final Schraube SCHRAUBE_4 = new Schraube("Schraube 4", 16, 3.0);
    public static final Schraube SCHRAUBE_5 = new Schraube("Schraube 5", 18, 3.5);
    public static final Schraube SCHRAUBE_6 = new Schraube("Schraube 6", 20, 4.0);
    public static final Schraube SCHRAUBE_7 = new Schraube("Schraube 7", 22, 4.5);
    public static final Schraube SCHRAUBE_8 = new Schraube("Schraube 8", 24, 5.0);
    public static final Schraube SCHRAUBE_9 = new Schraube("Schraube 9", 26, 5.5);
    public static final Schraube SCHRAUBE_10 = new Schraube("Schraube 10", 28, 6.0);
    public static final Schraube SCHRAUBE_11 = new Schraube("Schraube 11", 30, 6.5);
    public static final Schraube SCHRAUBE_12 = new Schraube("Schraube 12", 32, 7.0);
    public static final Schraube SCHRAUBE_13 = new Schraube("Schraube 13", 34, 7.5);
    public static final Schraube SCHRAUBE_14 = new Schraube("Schraube 14", 36, 8.0);
    public static final Schraube SCHRAUBE_15 = new Schraube("Schraube 15", 38, 8.5);
    public static final Schraube SCHRAUBE_299 = new Schraube("Schraube 299", 608, 15.0);
    public static final Schraube SCHRAUBE_300 = new Schraube("Schraube 300", 610, 15.5);

    @Override
    public String toString() {
        return "Schraube{" +
            "name='" + name + '\'' +
            ", laenge=" + laenge +
            ", durchmesser=" + durchmesser +
            '}';
    }
}
```

Kann man das nicht kürzer schreiben?



# Die Zeit vor Enums – Problemlösung am Beispiel Schrauben und Nägel im Bauhaus

```
public class Schraube {
    private String name;
    private int laenge; // Länge in mm
    private double durchmesser; // Durchmesser in mm

    private Schraube(String name, int laenge, double durchmesser) {
        this.name = name;
        this.laenge = laenge;
        this.durchmesser = durchmesser;
    }

    public static final Schraube SCHRAUBE_1 = new Schraube("Schraube 1", 10, 1.5);
    public static final Schraube SCHRAUBE_2 = new Schraube("Schraube 2", 12, 2.0);
    public static final Schraube SCHRAUBE_3 = new Schraube("Schraube 3", 14, 2.5);
    public static final Schraube SCHRAUBE_4 = new Schraube("Schraube 4", 16, 3.0);
    public static final Schraube SCHRAUBE_5 = new Schraube("Schraube 5", 18, 3.5);
    public static final Schraube SCHRAUBE_6 = new Schraube("Schraube 6", 20, 4.0);
    public static final Schraube SCHRAUBE_7 = new Schraube("Schraube 7", 22, 4.5);
    public static final Schraube SCHRAUBE_8 = new Schraube("Schraube 8", 24, 5.0);
    public static final Schraube SCHRAUBE_9 = new Schraube("Schraube 9", 26, 5.5);
    public static final Schraube SCHRAUBE_10 = new Schraube("Schraube 10", 28, 6.0);
    public static final Schraube SCHRAUBE_11 = new Schraube("Schraube 11", 30, 6.5);
    public static final Schraube SCHRAUBE_12 = new Schraube("Schraube 12", 32, 7.0);
    public static final Schraube SCHRAUBE_13 = new Schraube("Schraube 13", 34, 7.5);
    public static final Schraube SCHRAUBE_14 = new Schraube("Schraube 14", 36, 8.0);
    public static final Schraube SCHRAUBE_15 = new Schraube("Schraube 15", 38, 8.5);
    public static final Schraube SCHRAUBE_299 = new Schraube("Schraube 299", 608, 15.0);
    public static final Schraube SCHRAUBE_300 = new Schraube("Schraube 300", 610, 15.5);

    @Override
    public String toString() {
        return "Schraube{" +
            "name='" + name + '\'' +
            ", laenge=" + laenge +
            ", durchmesser=" + durchmesser +
            '}';
    }
}
```

Kann man das nicht kürzer schreiben?

DOCH! Mit Enums



# Die Zeit der Enums – Problemlösung am Beispiel Schrauben und Nägel im Bauhaus

```
enum Schraube {
```

```
SCHRAUBE_1("Schraube 1", 10, 1.5),
SCHRAUBE_2("Schraube 2", 12, 2.0),
SCHRAUBE_3("Schraube 3", 14, 2.5),
SCHRAUBE_4("Schraube 4", 16, 3.0),
SCHRAUBE_5("Schraube 5", 18, 3.5),
SCHRAUBE_6("Schraube 6", 20, 4.0),
SCHRAUBE_7("Schraube 7", 22, 4.5),
SCHRAUBE_8("Schraube 8", 24, 5.0),
SCHRAUBE_9("Schraube 9", 26, 5.5),
SCHRAUBE_10("Schraube 10", 28, 6.0),
SCHRAUBE_11("Schraube 11", 30, 6.5),
SCHRAUBE_12("Schraube 12", 32, 7.0),
SCHRAUBE_13("Schraube 13", 34, 7.5),
SCHRAUBE_14("Schraube 14", 36, 8.0),
SCHRAUBE_15("Schraube 15", 38, 8.5),
SCHRAUBE_299("Schraube 299", 608, 150.0),
SCHRAUBE_300("Schraube 300", 610, 150.5);
```

Sieht schon mal ordentlicher aus

```
private String name;
private int laenge; // Länge in mm
private double durchmesser; // Durchmesser in mm
```

```
private Schraube(String name, int laenge, double durchmesser) {
    this.name = name;
    this.laenge = laenge;
    this.durchmesser = durchmesser;
}
```

```
@Override
public String toString() {
    return "Schraube{" +
        "name=" + name + "\n" +
        "laenge=" + laenge +
        "durchmesser=" + durchmesser +
        '\n';
}
```

# Die Zeit der Enums – Problemlösung an unserem Beispiel Pizzagroesse

## Zuvor:

```
public class Pizzagroesse {  
    private String groesse;  
  
    private Pizzagroesse(String groesse) {  
        this.groesse = groesse;  
    }  
  
    public static final Pizzagroesse KLEIN = new Pizzagroesse("Klein");  
    public static final Pizzagroesse MITTEL = new Pizzagroesse("Mittel");  
}
```



# Die Zeit der Enums – Problemlösung an unserem Beispiel Pizzagroesse

## Zuvor:

```
public class Pizzagroesse {  
    private String groesse;  
  
    private Pizzagroesse(String groesse) {  
        this.groesse = groesse;  
    }  
  
    public static final Pizzagroesse KLEIN = new Pizzagroesse("Klein");  
    public static final Pizzagroesse MITTEL = new Pizzagroesse("Mittel");  
}
```

=

```
public enum Pizzagroesse {  
    KLEIN("Klein"),  
    MITTEL("Mittel");  
  
    private final String name;  
  
    Pizzagroesse(String name){  
        this.name = name;  
    }  
}
```

# Die Zeit der Enums – Problemlösung an unserem Beispiel Pizzagroesse

## Zuvor:

```
public class Pizzagroesse {  
    private String groesse;  
  
    private Pizzagroesse(String groesse) {  
        this.groesse = groesse;  
    }  
  
    public static final Pizzagroesse KLEIN = new Pizzagroesse("Klein");  
    public static final Pizzagroesse MITTEL = new Pizzagroesse("Mittel");  
}
```

=

```
public enum Pizzagroesse {  
    KLEIN("Klein"),  
    MITTEL("Mittel");  
  
    private final String name;  
  
    Pizzagroesse(String name){  
        this.name = name;  
    }  
}
```



# Aufbau komplexer Enums

```
public enum Pizzagroesse {
    KLEIN("Klein"),
    MITTEL("Mittel");

    private final String name;

    Pizzagroesse(String name){
        this.name = name;
    }

    ....
}
```

1. Schlüsselwort: enum, no- oder public-modifier

2. Konstanten:

Großgeschrieben nach  
Java-Konventionen

3. Semikolon notwendig,  
wenn mehr als nur Konstanten

4. Variablen/Eigenschaften

5. Konstruktor: ist und muss private sein

Optional : 6. Weitere Konstruktoren,  
Getter- und Setter-Methoden sowie  
weitere Methoden

# Aufbau verstanden?

```
enum Pizzagroesse {
    KLEIN("Klein", 26, 5.99),
    MITTEL("Mittel", 28, 8.99),
    GROSS("Groß", 30, 10.99),
    PARTY("Party", 60, 30.99)
```

```
    private final String name;
    private final int durchmesser;
    private final double preis;
```

```
    Pizzagroesse(String name, int durchmesser, double preis) {
        this.name = name;
        this.durchmesser = durchmesser;
        this.preis = preis;
    }
}
```

1. Schlüsselwort: enum, no- oder public-modifier
2. Konstanten: Großgeschrieben nach Java-Konventionen
3. Semikolon notwendig, wenn mehr als nur Konstanten
4. Variablen/Eigenschaften
5. Konstruktor: ist und muss private sein
- Optional: 6. Weitere Konstruktoren, Getter- und Setter-Methoden sowie weitere Methoden



# Aufbau verstanden?

```
enum Pizzagroesse {  
    KLEIN("Klein", 26, 5.99),  
    MITTEL("Mittel", 26, 8.99),  
    GROSS("Groß", 28, 10.99),  
    PARTY("Party", 60, 30.99)
```

→ Semikolon fehlt

```
    private final String name;  
    private final int durchmesser;  
    private final double preis;
```

```
    Pizzagroesse(String name, int durchmesser, double preis) {  
        this.name = name;  
        this.durchmesser = durchmesser;  
        this.preis = preis;  
    }  
}
```



# Aufbau verstanden?

```
enum Pizzagroesse {
    KLEIN("Klein", 26, 5.99),
    MITTEL("Mittel", 26, 8.99),
    GROSS("Groß", 28, 10.99),
    PARTY("Party", 60, 30.99);
```

```
    private final String name;
    private final int durchmesser;
    private final double preis;
```

```
    public Pizzagroesse(String name, int durchmesser, double preis) {
        this.name = name;
        this.durchmesser = durchmesser;
        this.preis = preis;
    }
}
```

1. Schlüsselwort: enum, no- oder public-modifier
2. Konstanten: Großgeschrieben nach Java-Konventionen
3. Semikolon notwendig, wenn mehr als nur Konstanten
4. Variablen/Eigenschaften
5. Konstruktor: ist und muss private sein
- Optional: 6. Weitere Konstruktoren, Getter- und Setter-Methoden sowie weitere Methoden



# Aufbau verstanden?

```
enum Pizzagroesse {  
    KLEIN("Klein", 26, 5.99),  
    MITTEL("Mittel", 26, 8.99),  
    GROSS("Groß", 28, 10.99),  
    PARTY("Party", 60, 30.99)  
  
    private final String name;  
    private final int durchmesser;  
    private final double preis;  
  
    public Pizzagroesse(String name, int durchmesser, double preis) {  
        this.name = name;  
        this.durchmesser = durchmesser;  
        this.preis = preis;  
    }  
}
```

Darf nicht public sein



# Aufbau verstanden?

```
enum Pizzagroesse {
    KLEIN("Klein", 26, 5.99),
    MITTEL("Mittel", 26, 8.99),
    GROSS("Groß", 28, 10.99),
    PARTY("Party", 60, 30.99);

    private final String name;
    private final int durchmesser;
    private final double preis;
}
```

1. Schlüsselwort: enum, no- oder public-modifier
2. Konstanten: Großgeschrieben nach Java-Konventionen
3. Semikolon notwendig, wenn mehr als nur Konstanten
4. Variablen/Eigenschaften
5. Konstruktor: ist und muss private sein
- Optional: 6. Weitere Konstruktoren, Getter- und Setter-Methoden sowie weitere Methoden



# Aufbau verstanden?

```
enum Pizzagroesse {  
    KLEIN("Klein", 26, 5.99),  
    MITTEL("Mittel", 26, 8.99),  
    GROSS("Groß", 28, 10.99),  
    PARTY("Party", 60, 30.99);
```

```
    private final String name;  
    private final int durchmesser;  
    private final double preis;
```

```
    Pizzagroesse(String name, int durchmesser, double preis) {  
        this.name = name;  
        this.durchmesser = durchmesser;  
        this.preis = preis;  
    }  
}
```

Konstruktor notwendig, wenn  
hinter den Konstanten was steht:  
automatischer Konstruktoraufruf



# Aufbau verstanden?

```
enum Pizzagroesse {
    KLEIN("Klein", 26, 5.99),
    MITTEL("Mittel", 26, 8.99),
    GROSS,
    PARTY("Party", 60);

    private String name;
    private int durchmesser;
    private double preis;

    Pizzagroesse(String name, int durchmesser, double preis) {
        this.name = name;
        this.durchmesser = durchmesser;
        this.preis = preis;
    }

    Pizzagroesse(String name, int durchmesser) {
        this.name = name;
        this.durchmesser = durchmesser;
    }

    Pizzagroesse() {
    }
}
```

1. Schlüsselwort: enum, no- oder public-modifier
2. Konstanten: Großgeschrieben nach Java-Konventionen
3. Semikolon notwendig, wenn mehr als nur Konstanten
4. Variablen/Eigenschaften
5. Konstruktor: ist und muss private sein
- Optional: 6. Weitere Konstruktoren, Getter- und Setter-Methoden sowie weitere Methoden





# Aufbau verstanden?

```
abstract enum Pizzagroesse {
    KLEIN("Klein", 26, 5.99),
    MITTEL("Mittel", 28, 8.99);

    private String name;
    private int durchmesser;
    private double preis;

    Pizzagroesse(String name, int durchmesser, double preis) {
        this.name = name;
        this.durchmesser = durchmesser;
        this.preis = preis;
    }
}
```

1. Schlüsselwort: enum, no- oder public-modifier
2. Konstanten: Großgeschrieben nach Java-Konventionen
3. Semikolon notwendig, wenn mehr als nur Konstanten
4. Variablen/Eigenschaften
5. Konstruktor: ist und muss private sein

Optional: 6. Weitere Konstruktoren, Getter- und Setter-Methoden sowie weitere Methoden



# Aufbau verstanden?

```
abstract enum Pizzagroesse {  
    KLEIN("Klein", 26, 5.99),  
    MITTEL("Mittel", 26, 8.99);  
}
```

Enum darf nicht als static, final, abstract, protected oder private deklariert werden

```
private String name;  
private int durchmesser;  
private double preis;
```

```
Pizzagroesse(String name, int durchmesser, double preis) {  
    this.name = name;  
    this.durchmesser = durchmesser;  
    this.preis = preis;  
}  
}
```



# Zusammenfassung Aufbau komplexer Enums

```
public enum Pizzagroesse {  
    KLEIN("Klein", 26, 5.99),  
    MITTEL("Mittel", 28, 8.99);
```

4. Konstanten müssen den Konstruktor „aufrufen“ & Werte zuweisen

```
    private final String name;  
    private final int durchmesser;  
    private final double preis;
```

1. Eigenschaften/ Variablen definieren

```
    Pizzagroesse(String name, int durchmesser, double preis){  
        this.name = name;  
        this.durchmesser = durchmesser;  
        this.preis = preis;  
    }
```

2. Konstruktor muss private sein

3. Konstruktor muss Attribute als Parameter entgegennehmen

```
    ...  
}
```

# Aufbau von einfachen Enums

- Anderes Beispiel – Pizzagroesse hat keine Eigenschaften (Grundpreis, Durchmesser, Name,...)

```
class Pizzagroesse {  
    public static final Pizzagroesse KLEIN = new Pizzagroesse();  
    public static final Pizzagroesse MITTEL = new Pizzagroesse();  
    public static final Pizzagroesse GROSS = new Pizzagroesse();  
}
```

Wie würde das als Enum aussehen?

# Aufbau von einfachen Enums

- Einfach, weil Objekte ohne Werte initialisiert werden

```
public enum Pizzagroesse {  
    KLEIN,  
    MITTEL,  
    GROSS  
}
```

# Aufbau von einfachen Enums

- Einfach, weil Objekte ohne Werte initialisiert werden

```
public enum Pizzagroesse {  
    KLEIN,  
    MITTEL,  
    GROSS  
}
```

1. Schlüsselwort: enum, nur no- oder public-modifier

2. Konstanten: Großgeschrieben nach Java-Konventionen

3. Semikolon optional, da keine  
Abtrennung zum Code notwendig

# Aufbau verstanden?

```
class Pizzagroesse {  
    KLEIN,  
    MITTEL,  
    GROSS,  
    PARTY;  
}
```

1. Schlüsselwort: enum, nur no- oder public-modifier
2. Konstanten: Großgeschrieben nach Java-Konventionen
3. Semikolon optional, da keine Abtrennung zum Code notwendig



# Aufbau verstanden?

```
enum Pizzagroesse {  
    KLEIN,  
    MITTEL,  
    GROSS,  
    PARTY;  
}
```





# Aufbau verstanden?

```
private enum Pizzagroesse {  
    KLEIN,  
    MITTEL,  
    GROSS,  
    PARTY  
}
```

1. Schlüsselwort: enum, nur no- oder public-modifier
2. Konstanten: Großgeschrieben nach Java-Konventionen
3. Semikolon optional, da keine Abtrennung zum Code notwendig



# Aufbau verstanden?

```
private enum Pizzagroesse {  
    KLEIN,  
    MITTEL,  
    GROSS,  
    PARTY  
}
```



# Aufbau verstanden?

```
public enum Pizzagroesse {  
    KLEIN,  
    MITTEL,  
    GROSS,  
    PARTY;  
}
```

1. Schlüsselwort: enum, nur no- oder public-modifier
2. Konstanten: Großgeschrieben nach Java-Konventionen
3. Semikolon optional, da keine Abtrennung zum Code notwendig



# Zusammenfassung Aufbau einfacher Enums

```
public enum Pizzagroesse {  
    KLEIN,  
    MITTEL  
}
```

1. enum statt class
2. Nur public/default – Access-Modifier legal
3. Großgeschriebene Konstanten
4. Mittels Komma getrennt
5. Semikolon am Ende optional

# Wie wird das Pizzagroessen-Enum verwendet?

```
public enum Pizzagroesse {  
    KLEIN("Klein"),  
    MITTEL("Mittel");  
  
    private final String name;  
  
    Pizzagroesse(String name){  
        this.name = name;  
    }  
}
```

```
import java.util.List;  
  
public class Pizza {  
  
    private Pizzagroesse pizzagroesse;  
    private List<Zutat> zutaten;  
  
    public Pizza(Pizzagroesse pizzagroesse, List<Zutat> zutaten){  
        this.pizzagroesse = pizzagroesse;  
        this.zutaten = zutaten;  
    }  
}
```

## Wie erstellen wir ein Pizza-Objekt?



# Wie wird das Pizzagroessen-Enum verwendet?

- Wie zuvor! Mittels Konstante

```
public enum Pizzagroesse {  
    KLEIN("Klein"),  
    MITTEL("Mittel");  
  
    private final String name;  
  
    Pizzagroesse(String name){  
        this.name = name;  
    }  
}
```

```
import java.util.List;  
  
public class Pizza {  
  
    private Pizzagroesse pizzagroesse;  
    private List<Zutat> zutaten;  
  
    public Pizza(Pizzagroesse pizzagroesse, List<Zutat> zutaten){  
        this.pizzagroesse = pizzagroesse;  
        this.zutaten = zutaten;  
    }  
}
```

```
Pizza pizza = new Pizza(Pizzagroesse.KLEIN, zutatenListe);
```

# Drei wichtige Enum-Methoden

# Drei wichtige Enum-Methoden

- **toString()**
  - holt sich, wenn nicht überschrieben, den Konstantennamen (z.B. KLEIN)
  - kann überschrieben werden, sodass bspw. der String zurückgegeben wird („Klein“)



# Drei wichtige Enum-Methoden

- **toString()**
  - holt sich, wenn nicht überschrieben, den Konstantennamen (z.B. KLEIN)
  - kann überschrieben werden, sodass bspw. der String zurückgegeben wird („Klein“)

```
public enum PizzaSize {  
  
    KLEIN("Klein", 26, 8.0),  
    MITTEL("Mittel", 30, 10.0),  
    GROSS("Groß", 32, 12.0),  
    FAMILIE("Familie", 40, 20.0),  
    PARTY("Party", 60, 40.0);  
  
    private int durchmesser;  
    private String name;  
    private double grundpreis;  
  
    PizzaSize(String name, int durchmesser, double grundpreis){  
        this.name = name;  
        this.durchmesser = durchmesser;  
        this.grundpreis = grundpreis;  
    }  
}
```

# Drei wichtige Enum-Methoden

- **toString()**
  - holt sich, wenn nicht überschrieben, den Konstantennamen (z.B. KLEIN)
  - kann überschrieben werden, sodass bspw. der String zurückgegeben wird („Klein“)

```
public enum PizzaSize {  
  
    KLEIN("Klein", 26, 8.0),  
    MITTEL("Mittel", 30, 10.0),  
    GROSS("Groß", 32, 12.0),  
    FAMILIE("Familie", 40, 20.0),  
    PARTY("Party", 60, 40.0);  
  
    private int durchmesser;  
    private String name;  
    private double grundpreis;  
  
    PizzaSize(String name, int durchmesser, double grundpreis){  
        this.name = name;  
        this.durchmesser = durchmesser;  
        this.grundpreis = grundpreis;  
    }  
}
```

```
public static void main(String[] args) {  
    System.out.println(Pizzagroesse.KLEIN.toString());  
}  
//Ausgabe: KLEIN
```

# Drei wichtige Enum-Methoden

- **toString()**
  - holt sich, wenn nicht überschrieben, den Konstantennamen (z.B. KLEIN)
  - kann überschrieben werden, sodass bspw. der String zurückgegeben wird („Klein“)

```
public enum PizzaSize {
```

```
    KLEIN("Klein", 26, 8.0),  
    MITTEL("Mittel", 30, 10.0),  
    GROSS("Groß", 32, 12.0),  
    FAMILIE("Familie", 40, 20.0),  
    PARTY("Party", 60, 40.0);
```

```
    private int durchmesser;  
    private String name;  
    private double grundpreis;
```

```
    PizzaSize(String name, int durchmesser, double grundpreis){  
        this.name = name;  
        this.durchmesser = durchmesser;  
        this.grundpreis = grundpreis;  
    }
```

```
    public String toString() {  
        return name + " \u2300" + durchmesser + "cm: " + grundpreis + "€";  
    }
```

//Beispielausgabe: Klein ø 26cm: 8.00 €

# Drei wichtige Enum-Methoden

- **toString()**
  - holt sich, wenn nicht überschrieben, den Konstantennamen (z.B. KLEIN)
  - kann überschrieben werden, sodass bspw. der String zurückgegeben wird („Klein“)
- **values()**
  - erstellt einen neuen Array mit allen existierenden Konstanten
  - -> [KLEIN, MITTEL, GROß, FAMILY, PARTY]

```
for(Pizzagroesse p : Pizzagroesse.values()){  
    System.out.println(p);  
  
}
```

# Drei wichtige Enum-Methoden

- **toString()**
  - holt sich, wenn nicht überschrieben, den Konstantennamen (z.B. KLEIN)
  - kann überschrieben werden, sodass bspw. der String zurückgegeben wird („Klein“)
- **values()**
  - erstellt einen neuen Array mit allen existierenden Konstanten
  - -> [KLEIN, MITTEL, GROß, FAMILY, PARTY]
- **valueOf(String s)**
  - holt sich das Objekt/Konstante, deren Name mit dem String s übereinstimmt
  - Bsp:
    - `Pizzagroesse groesse = Pizzagroesse.valueOf(„KLEIN“);`
    - Enum-Objekt KLEIN wird geholt

# Gängige Anwendungsfälle für Enums

- Sehr gut für switch-case nutzbar:

p ist eine beliebige Konstante des Enums Pizzagroesse.

```
switch(p){  
    case KLEIN:  
        System.out.println(„Klein“);  
        break;  
    case MITTEL:  
        System.out.println("Mittel");  
        break;  
    case GROSS:  
        System.out.println("Groß");  
        break;  
}
```

# Gängige Anwendungsfälle für Enums

Sehr gut für Schleifen nutzbar:

```
public enum Pizzagroesse {  
    KLEIN("Klein"),  
    MITTEL("Mittel");  
  
    private final String name;  
  
    Pizzagroesse(String name){  
        this.name = name;  
    }  
}  
  
public static void main(String[] args) {  
    for(Pizzagroesse p : Pizzagroesse.values()){  
        System.out.println(p);  
    }  
}
```

//->[KLEIN,MITTEL]

# Gängige Anwendungsfälle für Enums

Sehr gut für Schleifen nutzbar:

```
public enum Pizzagroesse {  
    KLEIN("Klein"),  
    MITTEL("Mittel");  
  
    private final String name;  
  
    Pizzagroesse(String name){  
        this.name = name;  
    }  
}  
  
public static void main(String[] args) {  
    for(Pizzagroesse p : Pizzagroesse.values()){  
        System.out.println(p);  
    }  
}
```

```
//Ausgabe : Konstantennamen  
KLEIN  
MITTEL
```



# Zusammenfassung

- Enums sind Aufzählungen („Enumeration“) von Konstanten
- Enum darf nicht als `static`, `final`, `abstract`, `protected` oder `private` deklariert werden
- Enums können Konstruktoren, Methoden und Variablen haben
- Können mehr als nur ein Argument im Konstruktor aufnehmen
- Konstruktoren können überladen werden
- Konstruktoren können nicht direkt aufgerufen werden -> automatischer Aufruf
- `values()`, `valueOf(String s)`, `toString()` sind wichtige Methoden von Enums

# Übung

- **Erstellt einen Enum Role mit folgenden Eigenschaften:**
  - `name : String`
  - `accessRights : int` (accessRights-Skala: 0 keine – 3 höchste)
  - **Es sollen folgende Rollen geben:**
    - Admin
    - CEO
    - Employee
    - Guest
  - **Welche Rolle soll welche Zugriffsrechte haben?**
- **Erstellt eine Klasse User mit folgenden Eigenschaften:**
  - `username : String`
  - `password : String`
  - `roles : List<Role>`

Pause

SMART  
INDUSTRY  
CAMPUS

# Alternative Schreibweisen – innerhalb einer Klasse

```
public class Pizza {
```

Enum

```
enum Pizzagroesse{KLEIN, MITTEL, GROSS};
```

Semikolon

```
private Pizzagroesse pizzagroesse;  
private List<Zutaten> zutaten;
```

```
public Pizza(Pizzagroesse pizzagroesse, List<Zutaten> zutaten){  
    this.pizzagroesse = pizzagroesse;  
    this.zutaten = zutaten;  
}
```

Verwendung: Pizza-Klasse  
muss auch genannt werden

```
public static void main(String[] args) {  
    Pizza pizza = new Pizza(Pizza.Pizzagroesse.KLEIN, zutatenListe);  
}
```

# Alternative Schreibweisen – Konstantenspezifischer Klassenkörper

```
public enum Pizzagroesse {  
    KLEIN,  
    MITTEL {  
        @Override  
        public int getCalculation() {  
            //komplexe Berechnung  
            return 2;  
        }  
    },  
    GROSS;  
  
    public int getCalculation(){  
        //komplexe Berechnung  
        return 1;  
    }  
}
```

Methode getCalculation() der Klasse wird für die Konstante MITTEL überschrieben

# Alternative Schreibweisen – Konstantenspezifischer Klassenkörper

```
public enum Pizzagroesse {  
    KLEIN,  
    MITTEL {  
        @Override  
        public int getPrio() {  
            return 2;  
        }  
    },  
    GROSS;
```

```
    public int getPrio(){  
        return 1;  
    }  
}
```

Methode getPrio() der Klasse wird für die Konstante MITTEL überschrieben

Was ist die Ausgabe von Pizzagroesse.KLEIN.getPrio()?



# Alternative Schreibweisen – Konstantenspezifischer Klassenkörper

```
public enum Pizzagroesse {  
    KLEIN,  
    MITTEL {  
        @Override  
        public int getPrio() {  
            return 1;  
        }  
    },  
    GROSS;
```

```
    public int getPrio(){  
        return 2;  
    }  
}
```

Methode getPrio() der Klasse wird für die Konstante MITTEL überschrieben

Was ist die Ausgabe von Pizzagroesse.MITTEL.getPrio()?



# Zusammenfassung

- Enums sind Aufzählungen („Enumeration“) von Konstanten
- Enum außerhalb einer Klasse darf nicht als `static`, `final`, `abstract`, `protected` oder `private` deklariert werden
- Enums können Konstruktoren, Methoden, Variablen und konstantenspezifische Klassenkörper haben
- Können mehr als nur ein Argument im Konstruktor aufnehmen
- Konstruktoren können überladen werden
- Konstruktoren können nicht direkt aufgerufen werden
- Können innerhalb/außerhalb einer Klasse definiert werden.  
NICHT INNERHALB einer METHODE
- `values()`, `valueOf(String s)`, `toString()` sind wichtige Methoden von Enums