

Call by Reference/Value

```

package referenzdatentypen;

public class Start {
    public static void main(String[] args) {
        Aufgerufene aufgerufene = new Aufgerufene();

        int b = 1;
        aufgerufene.increment(b);

        System.out.println("b = " + b);

        IntWrapper iw = new IntWrapper(1);
        aufgerufene.increment(iw);

        System.out.println("Wert von iw = " + iw.getI());
    }
}
    
```

```

package referenzdatentypen;

public class Aufgerufene {
    public int increment(int a) {
        a = a+1;
        return a;
    }

    public IntWrapper increment(IntWrapper iw) {
        int a = iw.getI();
        a += 1;
        iw.setI(a);

        return iw;
    }
}
    
```

call by value

call by reference

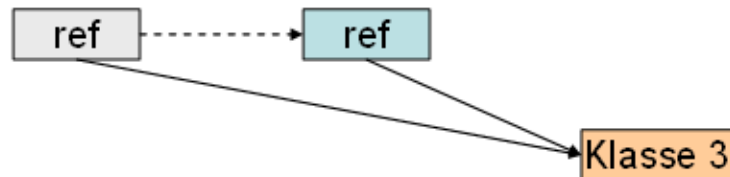
Start
Klasse 1

Aufgerufene
Klasse 2

Primitive Datentypen



Klassen



equals und ==

eigene equals()? Siehe nächste Seite

```
public static void main(String[] args) {
    Integer a = new Integer(42);
    Integer b = new Integer(42);

    System.out.println("a == b? " + (a == b)); //false
    System.out.println("a equals b? " + (a.equals(b))); //false, falls equals nicht überschrieben ist

    String c = new String("HTWG");
    String d = new String("HTWG");

    System.out.println("c == d? " + (c == d)); //false
    System.out.println("c equals d? " + (c.equals(d))); //true

    //String Literale
    String e = "HTWG";
    String f = "HTWG";

    System.out.println("e == f? " + (e == f)); //true |
    System.out.println("e equals f? " + (e.equals(f))); //true
}
```

Sonderfall!
Wert bei String-Literalen(!) hängt
von Implementierung der JVM ab.

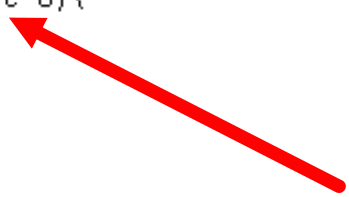
Implementieren von equals-Methoden

```
/**
 * überschreibt equals von Object
 */
public boolean equals(Object o){
    //1. Nullprüfung
    if (null == o){
        return false;
    }

    //2. Referenzprüfung
    if (o == this){
        return true;
    }

    //3. Prüfung, ob gleiche Klasse
    if (!(o instanceof IntWrapper)){
        return false;
    }

    //4. Individueller Vergleich der Attribute
    IntWrapper vergleich = (IntWrapper)o;
    if (vergleich.getI() == this.i){
        return true;
    }
    return false;
}
```



Object!!

Enums

```
public class Kalender {  
  
    public enum Jahreszeiten {Spring, Summer, Fall, Winter};  
  
    private Monat monat;  
    private Jahreszeiten jahrzeiten;  
  
    public Monat getMonat() {  
        return monat;  
    }  
  
    public void setMonat(Monat monat) {  
        this.monat = monat;  
    }  
  
    public Jahreszeiten getJahrzeiten() {  
        return jahrzeiten;  
    }  
  
    public void setJahrzeiten(Jahreszeiten jahrzeiten) {  
        this.jahrzeiten = jahrzeiten;  
    }  
}
```

interne, d.h. innerhalb einer Klasse definierte Enum

(externe) Enum, die in eigener Klasse definiert ist

```
public enum Monat {  
    Januar, Februar, Maerz  
}
```

Bitte beachten: Bei internen Enums, muss der Name der umgebenden/äußeren Klasse angegeben (bzw. importiert) werden!


```
public static void main(String[] args) {  
    Kalender kalender = new Kalender();  
    kalender.setMonat(Monat.Januar);  
    kalender.setJahrzeiten(Kalender.Jahreszeiten.Spring);  
}
```

Enums können über Attribute verfügen

```
package enums;

public enum Monat {
    Januar(31), Februar(28), Maerz(31);

    private int tage;
    private Monat(int tage) {
        this.tage = tage;
    }
}
```



Beachte, dass Konstruktor notwendig ist
(und Semikolon)

Enums können über Methoden verfügen

```
public enum Monat {  
    Januar {  
        String inEnglish(){  
            return "January";  
        }  
    },  
    Februar {  
        String inEnglish(){  
            return "February";  
        }  
    },  
    Maerz{  
        String inEnglish(){  
            return "March";  
        }  
    };  
  
    abstract String inEnglish();  
}
```

Abstrakte Methode ist optional, stellt aber sicher, dass jede der Enum-Werte die Methode implementiert und so von außen iteriert werden kann

```
for (Monat monat: Monat.values()) {  
    System.out.println("Monat " + monat +  
        " hat soviel Tage " + monat.getTage() +  
        " und der Monat heißt auf English " + monat.inEnglish());  
}
```

Beides (Attribute und Methoden) lässt sich kombinieren:

```
public enum Monat {  
    Januar(31){  
        String inEnglish(){  
            return "January";  
        }  
    },  
    Februar(28){  
        String inEnglish(){  
            return "February";  
        }  
    },  
    Maerz(31){  
        String inEnglish(){  
            return "January";  
        }  
    };  
  
    private Monat(int tage){  
        this.tage = tage;  
    }  
  
    public int getTage(){  
        return tage;  
    }  
  
    private final int tage;  
    abstract String inEnglish();  
}
```