Aufzählungstypen (enum)[¶](https://freiheit.f4.htw-berlin.de/prog2/enum/#aufzahlungstypen-enum)

Motivation[¶](https://freiheit.f4.htw-berlin.de/prog2/enum/#motivation)

Angenommen, Sie wollen mithilfe einer Variablen eine festgelegte Menge an Zuständen beschreiben, z.B.

String Tag = "MONTAG" // kann auch Werte "Dienstag" usw. annehmen

int Tag = 0; // Magic Number für "Montag"

Das Problem: - die Variablen können auch beliebige andere Werte (aus dem jeweiligen Wertebereich) annehmen, z.B. Tag = "hallo" oder Tag=4711, - Magic Numbers sollen vermieden werden → meistens schlechte Lesbarkeit

Beispiel TicTacToe

public class TicTacToe

{

int[][] field;

TicTacToe()

{

field = new int[3][3];

for(int i=0; i<field.length; i++)

for(int j=0; j<field[i].length; j++)

field[i][j]=0;

}

void makeMove(int i, int j, int player)

{

if(field[i][j]==0 && player==1 || player==2)

field[i][j]=player;

}

}

* Zustände EMPTY (0), RED (1), BLACK (2) verschlüsselt → magic numbers
* field[i][j] könnte auch beliebige andere int-Werte annehmen
* Code nahezu unlesbar

Erster Verbesserungsversuch: Konstanten[¶](https://freiheit.f4.htw-berlin.de/prog2/enum/#erster-verbesserungsversuch-konstanten)

Beispiel TicTacToe mit Konstanten

public class TicTacToe

{

int[][] field;

static final int EMPTY = 0;

static final int RED = 1;

static final int BLACK = 2;

TicTacToe()

{

field = new int[3][3];

for(int i=0; i<field.length; i++)

for(int j=0; j<field[i].length; j++)

field[i][j]=EMPTY;

}

void makeMove(int i, int j, int player)

{

if(field[i][j]==EMPTY && player==RED || player==BLACK)

field[i][j]=player;

}

}

* etwas besser, aber immer noch beliebige Werte für field[i][j] möglich

Der Aufzählungstyp enum[¶](https://freiheit.f4.htw-berlin.de/prog2/enum/" \l "der-aufzahlungstyp-enum" \o "Permanent link)

Anforderungen:

* eigener Datentyp
* endliche Anzahl an Zuständen bzw. Werten
* leserliche Bezeichnung der Werte

Lösung: - *Enumerations* (sog. Aufzählungstypen) - Schlüsselwort enum

Syntax:

enum TypName {WERT1, WERT2, WERT3};

Verwendung: - TypName nun als Datentyp verwendbar, z.B. TypName[][] - Zugriff auf Werte über statische Punktschreibweise, z.B. TypName.WERT1

Beispiel TicTacToe mit enum

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | public class TicTacToe  {  enum State {EMPTY, RED, BLACK};  State[][] field;  TicTacToe()  {  field = new State[3][3];  for(int i=0; i<field.length; i++)  for(int j=0; j<field[i].length; j++)  field[i][j]=State.EMPTY;  }  void makeMove(int i, int j, State player)  {  if(field[i][j]==State.EMPTY && player!=State.EMPTY)  field[i][j]=player;  }  } |

* typsicher
* rot und schwarz über State.RED und State.BLACK erreichbar
* andere Zustände nicht möglich

**Details:**[**¶**](https://freiheit.f4.htw-berlin.de/prog2/enum/#details)

* alle enum erben implizit von java.lang.Enum
* enum sind Referenztypen
* die Konstanten (Werte) in enum sind automatisch static und final
* == kann verwendet werden (auch switch()); equals() gibt es aber auch

Beispiel enum

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | State s = State.EMPTY; // s = 0 oder s = "rot" oder so geht nicht  //-> typsicher  switch(s)  {  case EMPTY: System.out.println("leeres Feld"); break;  case RED: System.out.println("roter Stein"); break;  case BLACK: System.out.println("schwarzer Stein Feld"); break;  } |

**Weiteres:**[**¶**](https://freiheit.f4.htw-berlin.de/prog2/enum/#weiteres)

* auch Definition von Methoden möglich
* toString(), equals() usw. aus Object können überschrieben werden
* Konstanten können mit Attributen versehen werden (dann noch privater Konstruktor notwendig)
* Zugriff auf das Array von Konstanten mithilfe von values()

Beispiel für Werte mit Attributen

enum Farben {

KREUZ(12), PIK(11), HERZ(10), KARO(9);

private int farbwert;

private Farben(int wert)

{

this.farbwert=wert;

}

@Override

public String toString()

{

char c = ' ';

switch(this)

{

case KREUZ : c ='\u2663'; break;

case PIK : c ='\u2664'; break;

case HERZ : c ='\u2665'; break;

case KARO : c ='\u2666'; break;

}

return String.valueOf(c);

}

}