Objektorientierte Programmierung mit C++ - Übungsserie 7

Rico Kölling 192316

```
1 Time t1(10, 10);
2 Time t2;
3 t2 = input();
4 t2.inc();
5 print(t2);
6 print(1234);
7 return 0;
```

Codezeile:	Konstruktor/Destruktor	Begründung
1	**Minuten-Stunden-Konstruktor**	Es werden zwei Werte an den Konstruktor übergeben, dieser wähl dann den Überladenen Minuten-Stunden-Konstruktor explizit, um das Element zu erschaffen.
2	**Standard-Konstruktor**	Bei t2 werden keine Parameter übergeben, somit wird der Standard-Konstruktor explizit aufgerufen.
3	**Standard-Konstruktor**	Beim Aufruf der Methode „input“ wird indirekt ein Objekt „Time result“ erschaffen welches den Standard-Konstruktor benutzt, weil keine Parameter übergeben werden.
3	**Destruktor**	Bei der Rückgabe von „input“, also „result“, wird der Destruktor indirekt aufgerufen, weil das Objekt „result“ seinen Gültigkeitsbereich verlässt.
5	**Kopier-Konstruktor**	Wenn „print(t2)“ aufgerufen wird, wird der Kopier-Konstruktor aufgerufen, weil „print“ ein Objekt bekommt.
5	**Destruktor**	In „print“ wird der Destruktor implizit am Ende der Methode aufgerufen, weil „t“ seinen Gültigkeitsbereich verlässt.
6	**Minuten-Konstruktor**	Es wird „print(1234)“ aufgerufen, dabei wird implizit in print das Objekt „t“ erstellt welches mit dem Wert: „1234“ ein Objekt bildet.
6	**Destruktor**	Das „t“ was vorher erschaffen wurde, wird jetzt durch das Verlassen seines Gültigkeitsbereich mit einem impliziten Aufruf des Destruktors destrukted.
7	**Destruktor**	„t1“ verlässt seinen Gültigkeitsbereich, weswegen der Destruktor implizit aufgerufen wird.
7	**Destruktor**	„t2“ verlässt seinen Gültigkeitsbereich, weswegen der Destruktor implizit aufgerufen wird.

Bemerkung: Mit Verlassen des Gültigkeitsbereiches meine ich, dass die lokal initialisierte Variable nur bis zum Ende der Funktion existiert. Das bedeutet sie wird am ende vom Destruktor zerstört.