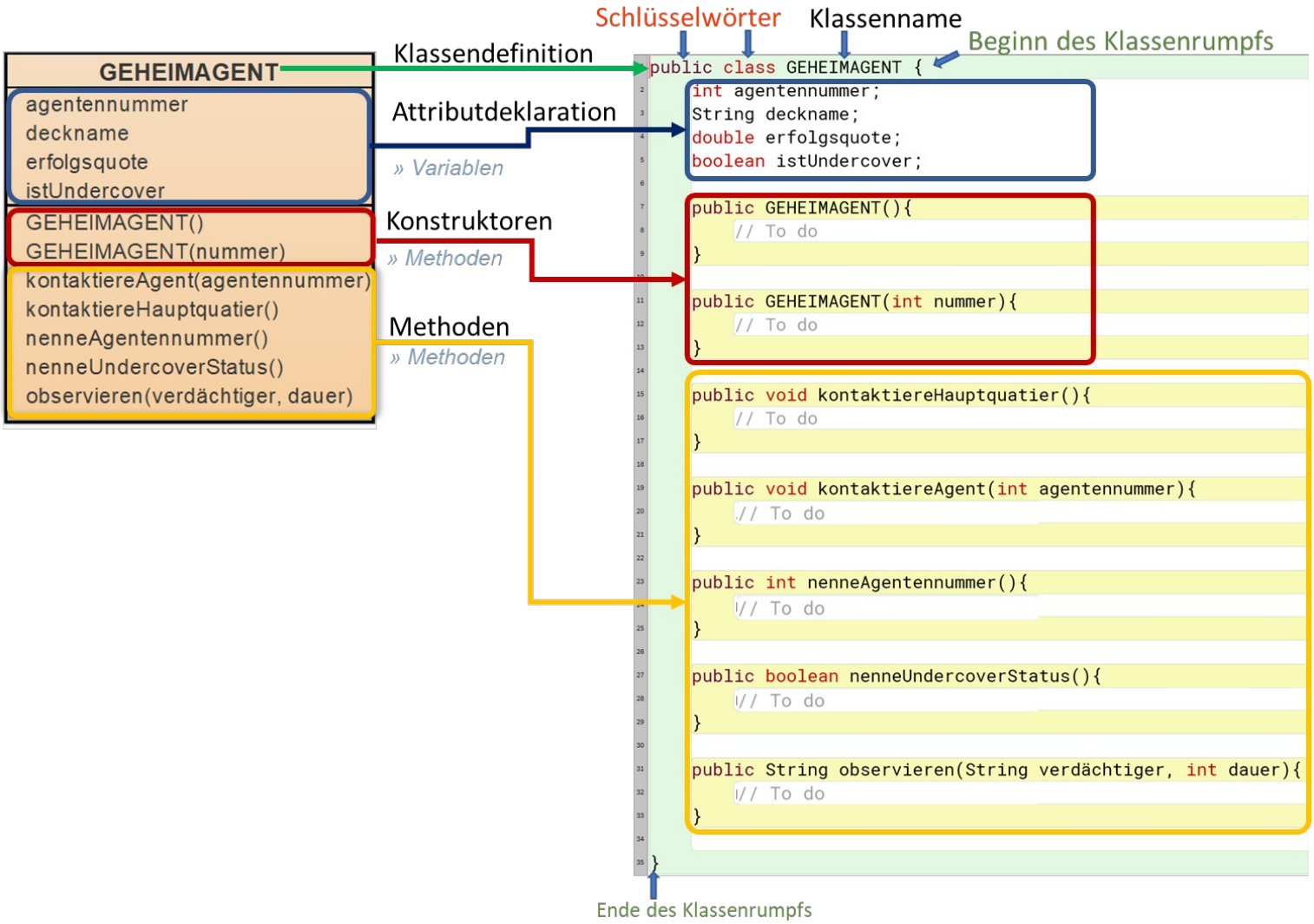
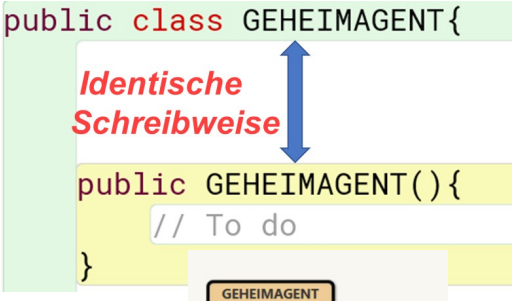
Klassenaufbau und Vererbung

Klassenaufbau



Eine ***Klasse*** definiert als Schablone die ***Attribute*** und ***Methoden***, die alle Objekte dieser Klasse haben sollen.

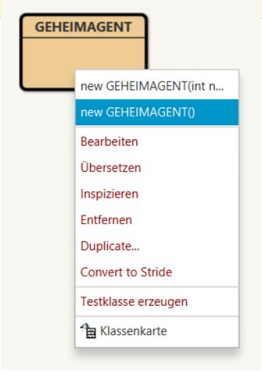
Konstruktor(-methoden)



Jede Klasse benötigt ***mindestens eine*** Konstruktormethode, damit Objekte dieser Klasse erzeugt werden können. Beim Aufruf der Konstruktormethode wird ein neues Objekt dieser Klasse mit den entsprechenden Attributwerten erstellt.

» weitergehende Informationen zur Erstellung von Attributen im Skript zu Variablen

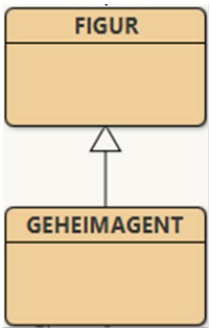
» weitergehende Informationen zur Erstellung von Methoden bzw. der Konstruktormethode im Skript zu Methoden



Vererbung: Spezialisierung (Erweiterung einer Klasse)

Eine Klasse kann von einer anderen Klasse mittels des Schlüsselworts ***extends*** erben.

In diesem Fall ist FIGUR die ***Oberklasse*** bzw. ***Superklasse*** und GEHEIMAGENT ist die ***Unterklasse*** bzw. ***Subklasse***.




```
public class GEHEIMAGENT extends FIGUR {

    public GEHEIMAGENT(){

    }

}
```

Die Klasse GEHEIMAGENT ***erbt alle Attribute und Methoden der Oberklasse FIGUR***. Damit lässt sich ein Geheimagent aus Figurteilen zusammensetzen.

Somit sind logischerweise Objekte der Klasse GEHEIMAGENT auch FIGUREn.

Obwohl in der Klasse GEHEIMAGENT noch kein (eigener) Inhalt implementiert wurde, besitzt das Objekt der Klasse GEHEIMAGENT schon viele geerbte Attribute und Methoden.

In der Unterklasse GEHEIMAGENT können ***eigene Attribute und Methoden*** implementiert werden und so die Eigenschaften und Fähigkeiten von GEHEIMAGENT erweitert werden. Dadurch haben dann die Objekte von GEHEIMAGENT mehr Eigenschaften und Fähigkeiten als eine gewöhnliche FIGUR und sind dadurch FIGUREN mit ***zusätzlichen spezielleren*** Attributen und Fähigkeiten von GEHEIMAGENTEN.

Überschreiben von Methoden

Die Klasse GEHEIMAGENT lässt sich auch weiter spezialisieren z. B. in die beiden Unterklassen NSA und MI6. Diese können dann wiederum um zusätzliche Attribute und Methoden erweitert werden. Bei der Spezialisierung kann es dann sein, dass Methoden der Oberklasse ***nicht mehr passend*** für die Objekte der Unterklasse sind und müssen ***daher angepasst werden***.

Beispiel:
Der Geheimagent hat die Fähigkeit sein Hauptquartier zu kontaktieren. Beide Objekte der Unterklassen können dies nun ebenfalls, aber der MI6-Agent möchte sein Hauptquartier in London anrufen, während der NSA-Agent sein Hauptquartier in Fort Mead anrufen muss. Somit ist die allgemeine Methode von Geheimagent nicht mehr passend für beide Unterklassen.

Das Anpassen der Methode kann durch das erneute Definieren der Methode in der Unterklasse erfolgen. Das nennt man ***Überschreiben*** (***Override***) der Methode, denn beim Aufruf der Methode wird dann ***immer die Methode der Unterklasse verwendet***. Dabei muss die Methode der Unterklasse exakt ***dieselbe Signatur*** haben wie die Methode der Oberklasse.

```
public class GEHEIMAGENT extends FIGUR{

    public void kontaktiereHauptquatier(){
        handy.baueverschlüsselteVerbindungauf("Hauptquatier");
        identifiziereMich();
    }
```

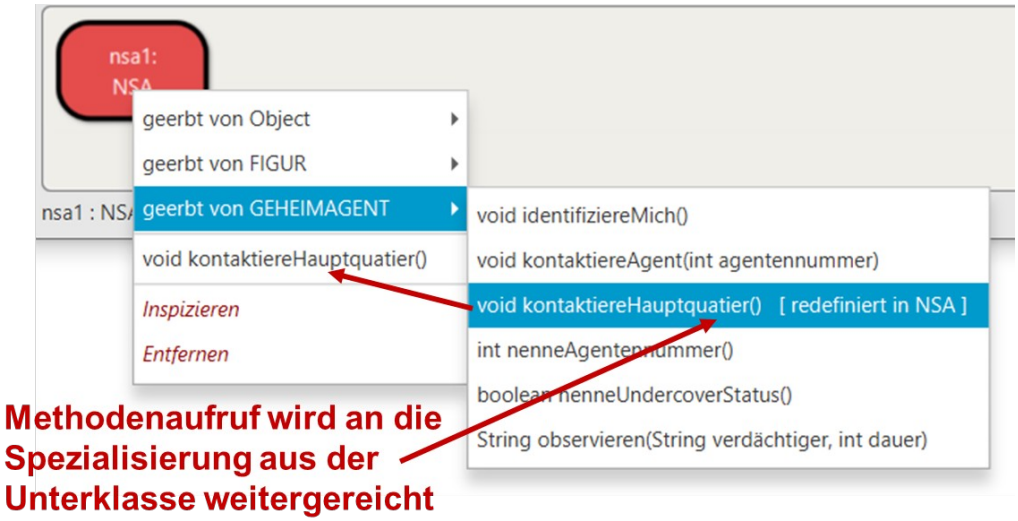
identische Signatur

@override-Kennzeichnung
kann auch weggelassen werden

```
public class MI6 extends GEHEIMAGENT{

    //@override
    public void kontaktiereHauptquatier(){
        handy.wechsleStealthModus();
        handy.baueverschlüsselteVerbindungauf("Hauptquatier");
        identifiziereMich();
        handy.übertrageSchlüssel("0x7A5EDA");
    }
```

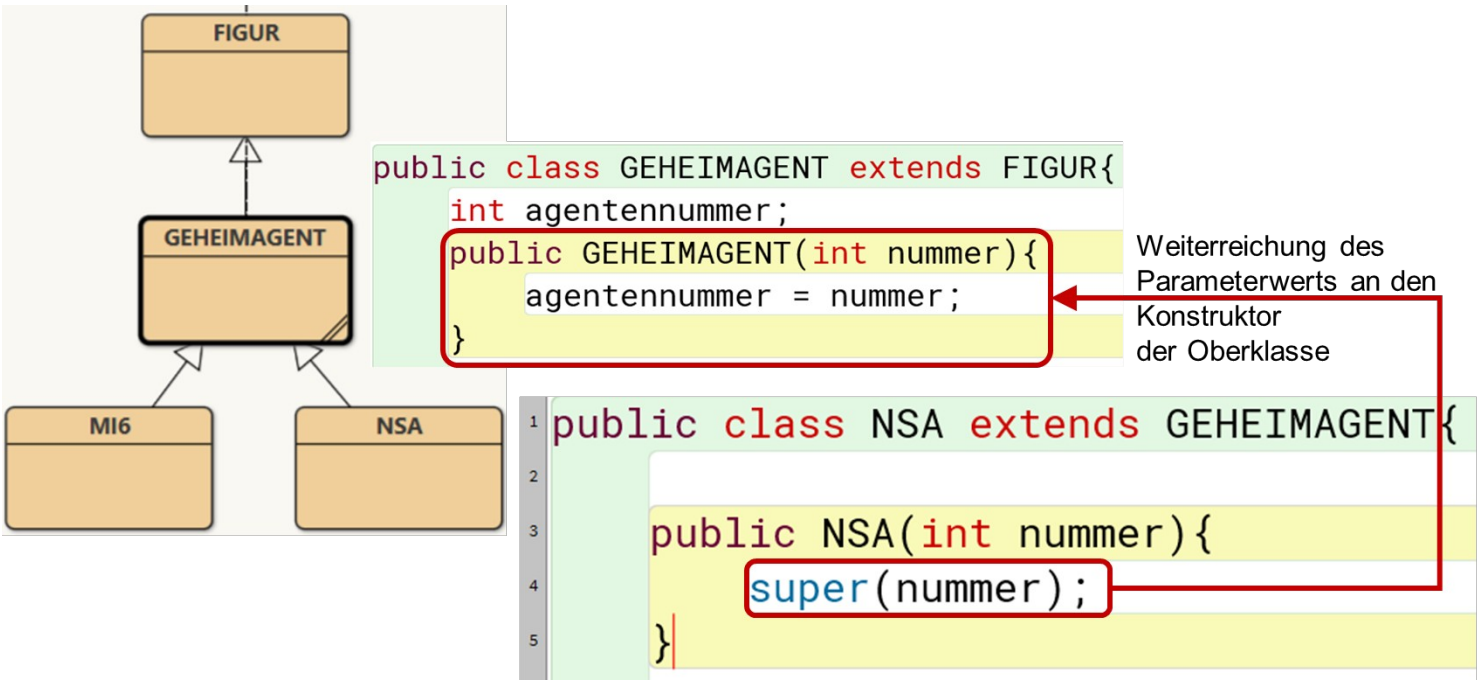
```
public class NSA extends GEHEIMAG

    //@override
    public void kontaktiereHauptq
        handy.baueverschlüsselteV
        identifiziereMich();
        handy.übertrageSchlüssel("0x93E4DA");
    }
```

Selbst wenn man versucht die Methode der Oberklasse auszuführen, wird dennoch die ***überschriebene Methode der Unterklasse ausgeführt***.



Kontruktormethoden innerhalb der Vererbung

Wird beim Erzeugen neuer Objekte der Unterklasse die Initialisierung der Attribute der Oberklasse benötigt (z. B. der Radius eines Kreises), so muss man ***als erste Anweisung*** innerhalb der Konstruktordefinition den passenden Konstruktor der **direkten Oberklasse** mittels des Schlüsselwortes **super(*Parameter*)** aufrufen.



Vererbung: Generalisierung (Zusammenfassen)

Beim Bearbeiten eines Projekts kann es vorkommen, dass man ***mehrere sehr ähnliche Klassen*** implementiert wie z. B. mehrere Spielfiguren oder Gegner. Hier kann man ebenfalls die Vererbung anwenden, um den Code besser zu strukturieren, indem man eine „abstraktere“ (allgemeinere) ***Oberklasse*** implementiert, die ***alle gemeinsamen Attribute und Methoden der ähnlichen Klassen vereint***. Dies nennt man ***Generalisierung***. Die ähnlichen Klassen (Unterklassen) erben wiederum von der neuen Oberklasse und haben damit dieselben Attribute und Fähigkeiten wie vorher. Aber man benötigt so insgesamt weniger Code, der besser strukturiert ist und damit auch weniger fehleranfällig ist.