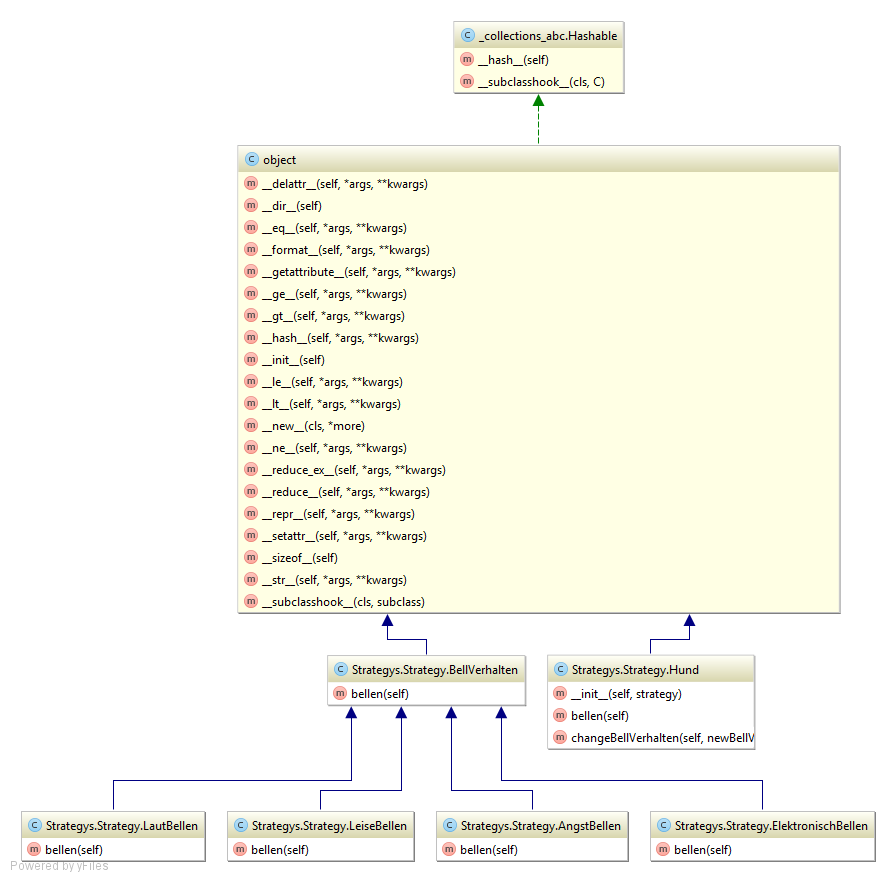
# Strategy

## UML



## Code:

\_\_author\_\_ **=** 'SECKIN Berkay'

# The strategy interface:

**class** **BellVerhalten** **:**

# Line is a sequence of points:

**def** bellen**(**self**)** **:** **pass**

"""

Beim definieren einer class kann man erben in dem man im Parameter den Namen dieser Klasse angibt.

Wird verwendet für attribut referenzen: Wenn ein benötigtes attribut nicht in dieser Klasse gefunden wird,

wird in der Basis Klasse(welche im Parameter übergeben wurde) gesucht.

Abgeleitete klassen können methoden von der Basis Klasse überschreiben

"""

# Verschiedene Strategys

**class** **LautBellen(**BellVerhalten**):**

**def** bellen**(**self**):**

**return** "GANZ LAUT BELLEN!!" # Dummy

**class** **LeiseBellen(**BellVerhalten**):**

**def** bellen**(**self**):**

**return** "ganz leise bellen..." # Dummy

**class** **ElektronischBellen(**BellVerhalten**):**

**def** bellen**(**self**):**

**return** "Elekkkkktronisch Bellen!" # Dummy

**class** **AngstBellen(**BellVerhalten**):**

**def** bellen**(**self**):**

**return** "bellen aus Angst!" # Dummy

# Diese Klasse ist für die Steuerung:

**class** **Hund:**

**def** \_\_init\_\_**(**self**,** strategy**):**

self**.**strategy **=** strategy

**def** bellen**(**self**):**

**return** self**.**strategy**.**bellen**()**

**def** changeBellVerhalten**(**self**,** newBellVerhalten**):**

self**.**strategy **=** newBellVerhalten

# Beispiel-Client

meinHund **=** Hund**(**LautBellen**())**

**print(**meinHund**.**bellen**())**

# Die Strategy ändern

meinHund**.**changeBellVerhalten**(**LeiseBellen**())**

**print(**meinHund**.**bellen**())**

## Das Strategy-Pattern in Python:

Das Strategy-Pattern wird verwendet um während der Laufzeit Änderungen durchführen zu können.

Man benötigt Interfaces, In Python aber gibt es diese nicht direkt, wie in Java. Jedoch gibt es als Abhilfe die Vererbung in Python(auch Mehrfach-Vererbung), welche folgende Syntax hat:

**class** **abgeleiteteKlasse(BasisKlasse):**

…  
 …

Dies wird verwendet für Attribut Referenzen: Wenn ein benötigtes Attribut nicht in dieser Klasse gefunden wird, wird in der Basis Klasse gesucht. In unserem Beispiel wird also das Verhalten(Funktion, Algorithmus) von dem Objekt Hund in eine eigene Strategyklasse ausgelagert.  
*Der Context(Hund-Klasse) hält eine Referenz auf sein Strategieobjekt und wenn er das ausgelagerte Verhalten ausführen soll, so delegiert er den Aufruf an sein referenziertes Strategieobjekt“ [3]*

# Command

## UMLC:\Users\Berkay\AppData\Local\Microsoft\Windows\INetCache\Content.Word\CommandPattern diagram.png

## Code

\_\_author\_\_ **=** 'Berkay'

"""

Mit dem Command-Design Pattern kann man bestimmte Aktionen einer Methode oder einem Objekt übergeben

=> Ein Befehl als Objekt kapseln:

Dies ermöglicht es, Klienten mit verschiedenen Anfragen zu parametrisieren,

Operationen in eine Schlange zu stellen, ein Logbuch zu führen und Operationen rückgängig zu machen.

"""

# Sozuasgen das Interface

**class** **DruckBefehl:**

**def** drucken**(**self**):** **pass**

# konkrete Durchführung

**class** **SchwarzWeissDruck(**DruckBefehl**):**

**def** drucken**(**self**):**

"""

druckt Schwarz/Weiss aus

:return: \_string

"""

**print(**"==== Schwarz/Weiß gedruckt ===="**)**

# konkrete Durchführung

**class** **FarbDruck(**DruckBefehl**):**

**def** drucken**(**self**):**

"""

druckt FARBIG aus

:return \_string:

"""

**print(**"==== FARBIG gedruckt ===="**)**

# konkrete Durchführung

**class** **PDFDruck(**DruckBefehl**):**

**def** drucken**(**self**):**

"""

druckt PDF aus

:return: \_string

"""

**print(**"==== PDF - Druck ===="**)**

# An object that holds commands:

**class** **Drucker:**

**def** \_\_init\_\_**(**self**):**

"""

Beim Aufruf dieser Klasse

:rtype : object

:return: None

"""

self**.**druckAuftraege **=** **[]** # leeres Array

**def** addAuftrag**(**self**,** druckAuftrag**):**

"""

Fügt ein Druckauftrag zur Auftragsliste hinzu

:param druckAuftrag:

:return: None

"""

self**.**druckAuftraege**.**append**(**druckAuftrag**)** # dem Array hinzufügen

**def** run**(**self**):**

"""

Durchführung aller Druckaufträge

:return:

"""

**for** c **in** self**.**druckAuftraege**:**

c**.**drucken**()**

**def** changeDruckAuftrag**(**self**):**

**pass**

# Beispiel - Client

meinDrucker **=** Drucker**()**

meinDrucker**.**addAuftrag**(**SchwarzWeissDruck**())**

meinDrucker**.**addAuftrag**(**FarbDruck**())**

meinDrucker**.**addAuftrag**(**PDFDruck**())**

meinDrucker**.**run**()**

# Das Command-Pattern:

Ein Objekt wird verwendet um all benötigten Informationen zu verkapseln und dann eine Methode aufrufen zu können. Mit dem Command-Design Pattern kann man bestimmte Aktionen einer Methode oder einem Objekt übergeben

=> Ein Befehl als Objekt kapseln:

Dies ermöglicht es, Klienten mit verschiedenen Anfragen zu parametrisieren,

Operationen in eine Schlange zu stellen, ein Logbuch zu führen und Operationen rückgängig zu machen.

*[1] Python Software Foundation; „Documentation >> The Python Tutorial >> 9.5. Ineritance“; Online:* [*https://docs.python.org/2/tutorial/classes.html*](https://docs.python.org/2/tutorial/classes.html)*; zuletzt aufgerufen: 21.12.2014; zuletzt geändert: 12.10.2014  
[2] „Strategy: Choosing the Algorithm at Runtime“; Online:* [*http://python-3-patterns-idioms-test.readthedocs.org/en/latest/FunctionObjects.html#strategy-choosing-the-algorithm-at-runtime*](http://python-3-patterns-idioms-test.readthedocs.org/en/latest/FunctionObjects.html#strategy-choosing-the-algorithm-at-runtime)*; zuletzt aufgerufen: 21.12.2014; zuletzt geändert: 10.12.2014*

*[3] Philipp Hauer; „Das Strategy Design Pattern“; Online:* [*http://www.philipphauer.de/study/se/design-pattern/strategy.php*](http://www.philipphauer.de/study/se/design-pattern/strategy.php)*; zuletzt aufgerufen: 21.12.2014; zuletzt geändert:*

*[4] Wikipedia; „Command Pattern“; Online:* [*http://en.wikipedia.org/wiki/Command\_pattern*](http://en.wikipedia.org/wiki/Command_pattern)*; zuletzt aufgerufen: 21.12.2014; zuletzt geändert: 15.12.2014*

*[5] Philip Hauer; „Das Command Design Pattern“; Online:* [*http://www.philipphauer.de/study/se/design-pattern/command.php*](http://www.philipphauer.de/study/se/design-pattern/command.php)*; zuletzt aufgerufen: 21.12.2014*

*[6] „Command: Choosing the Operation at Runtime“; Online:* [*http://python-3-patterns-idioms-test.readthedocs.org/en/latest/FunctionObjects.html#command-choosing-the-operation-at-runtime*](http://python-3-patterns-idioms-test.readthedocs.org/en/latest/FunctionObjects.html#command-choosing-the-operation-at-runtime)*; zuletzt aufgerufen: 21.12.2014; zuletzt geändert: 10.12.2014*