Bildungszentrum für Technik Frauenfeld

Thurgau

Tag 7

Einstieg OOP und Übung mit Git

31. März 2023

Bildet. Aus und Weiter.

Ablauf

- Rückblick Sechster Tag
- Übung ‚Formelparser' besprechen
- Einstieg OOP
  (Objekt-Orientierte Programmierung)
- Übung ‚Geometric Shapes'
  - Klassen erstellen
  - Instanzen erzeugen
  - Git anwenden: clonen, branchen, commiten

Rückblick Tag 6

Lernziele

- OOP kennen
- Können Vorteile von OOP aufzählen
- Klassen erstellen können
- Objekte erzeugen können
- Git clone, branch, commit, push, pull angewendet haben

Was bedeutet OOP?

- Es geht um **Objekte** und **Klassen**
  - Bisher hatten wir ‹nur› Funktionen und Daten
- Objekte/Klassen bündeln Funktionen und Daten
  - Daten werden geschützt, Zugriff erfolgt über Funktionen → Kapselung
- Abstraktion: Klassen beschreiben ein Konzept, ohne die Implementierung offenlegen zu müssen
- Polymorphie / Vererbung: Konzepte können generell beschrieben werden; Unterklassen spezialisieren dies Konzepte dann

  Hund → Hütehund → Border Collie

  Geometrie → geschlossene 2D Geometrie → Rechteck

Klasse ↔ Objekt

- **Klasse**: beschreibt das «Konzept» / «Model» → Idee, Datentyp
  z.B. Rechteck:
  Daten: Position, Länge, Breite
  Funktionen: move(), paint(), calculate_area(), calculate_circumference()

- **Objekt**: Eine Instanz von diesem Datentyp
  beliebig viele Instanzen einer einzigen Klasse
  z.B. Rechtecke: [‹@0/0 100x45›, ‹@200/299 10x20›, ‹@40/60 50x300›, ….]

Beispiel ***string***

- **Klasse** string
  - Konzept/Idee einer Zeichenkette
  - Daten nicht sichtbar → Kapselung
  - Zugriff nur über Funktionen
  - Wieviele? Beliebig - Keine. Viele.

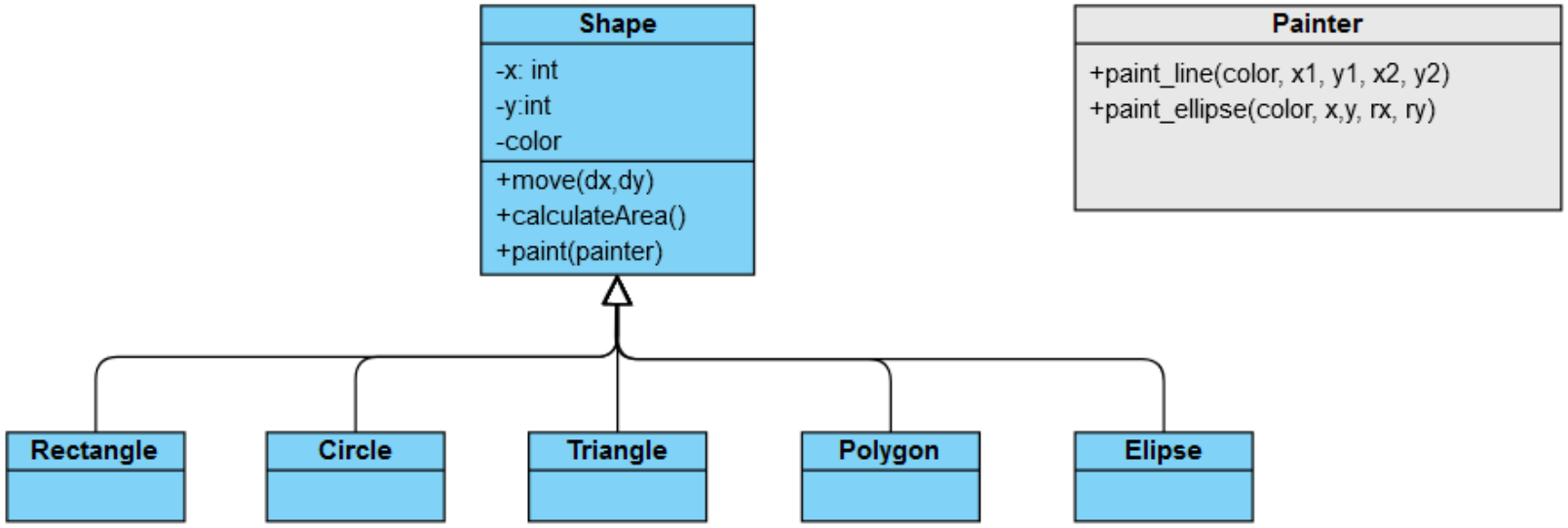
- **Objekt** name="Clau"
  Eine Instanz der Klasse string
  Der Datentyp von *name* ist string

Beispiel *dog_example.py*

```python
class Dog:
    """Base class for all dogs. Dogs have the properties 'name' and 'max_speed' and can 'say_name()'. """

    def __init__(self, name, max_speed):
        """Init, the constructor creates an instance of a dog. It requires the argument name and max_speed."""
        self._name = name
        self._max_speed = max_speed

    def say_name(self):
        """Lets the dog say its name."""
        print(f"My name is '{self._name}'")

    def get_name(self):
        """Returns the name of the dog."""
        return self._name

    def get_max_speed(self):
        """returns the max. speed this dog can reach."""
        return self._max_speed
```

Übung *Geometric Shapes*

Ziel: Gemeinsam entwickeln wir eine Klassenbibliothek von geometrischen Figuren



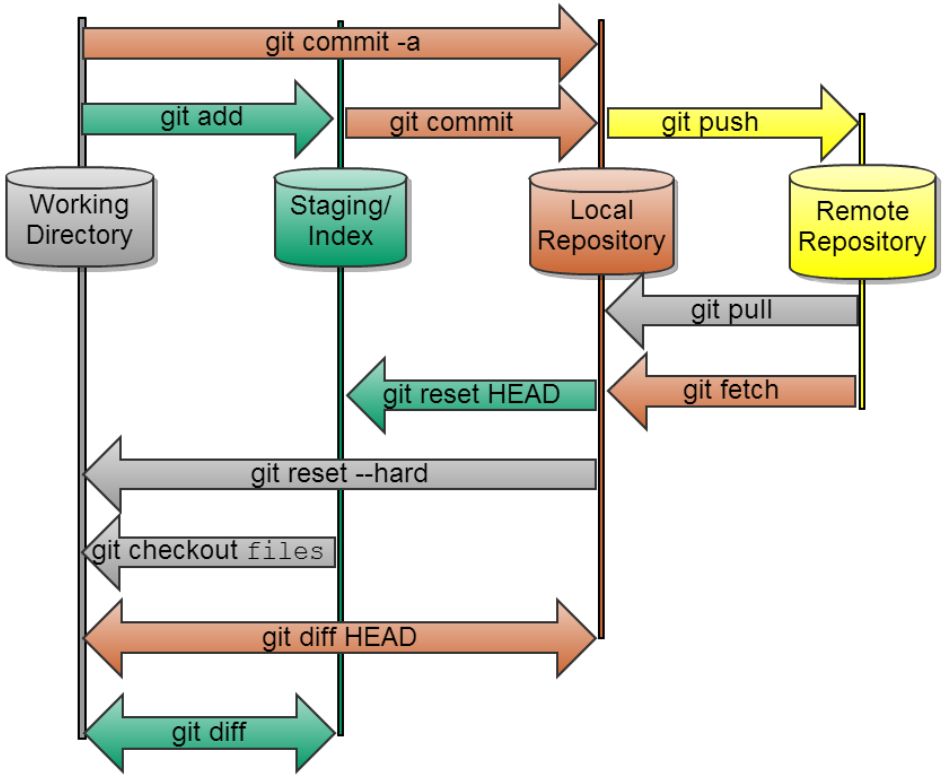
Übung *Geometric Shapes*

Ablauf:

1) Gruppen bilden, Shape-Klasse zuordnen
2) GitHub Account erstellen und *Git Graph* Plugin installieren
3) Git-Repository clonen (Angeleitet)
4) Feature-Branch erstellen (angeleitet)
5) Klasse implementieren durch Adaptation des Hunde-Beispiels
   - Klasse mit Daten und Funktionen *move()* und *calculate_area()*
   - Unit-test implementieren für funktion calculate_area()
     → Mehrmals auf den Feature-Branch commiten
6) Feature-Branch schlussendlich auf den Master mergen (angeleitet)

Git-Repo:
https://github.com/MicCalo/B06_GeometryExcercise

Git Übersicht



Git clone repository

Repository vom Server holen

- Öffne die Komandopallete (*Ctrl-Strg-P*) und wähle *Git: clone*
- Wähle *Git Repository*
- Gib *https://github.com/MicCalo/B06_GeometryExcercise* ein
- Wähle einen Ziel-Ordner
- Bestätige, dass dieser Zielordner geöffnet werden soll.

Git create branch

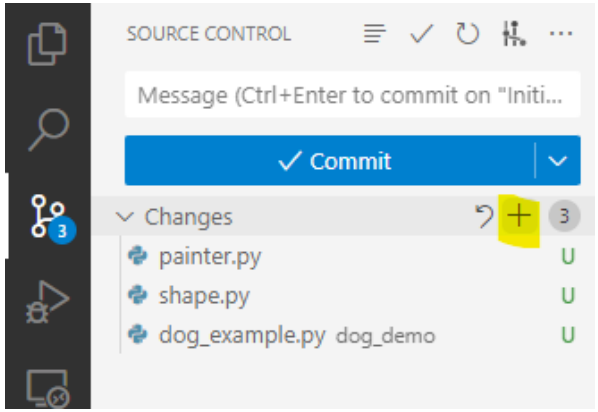
Einen Feature-Branch lokal erstellen, um ein Feature zu entwickeln, ohne den Main-Branch (Master) in Mitleidenschaft zu ziehen

- Komandopallete Git: *Create Branch…*
- Branch-Name eingeben (Feature-Name, z.b. AddPolygonShape)
- In der Source-Control-Ansicht () «Publish Branch» wählen

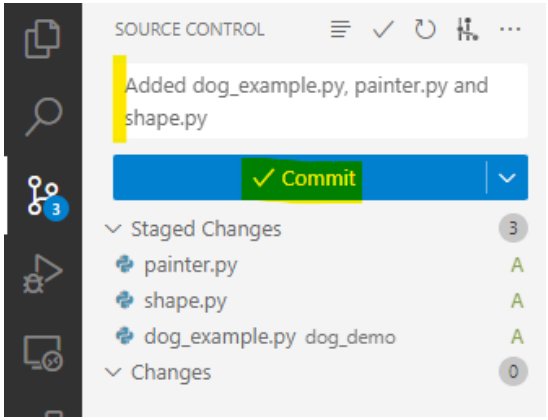
Git commit & push

Änderungen im Repository bestätigen (lokal: commit und auf den server:push)

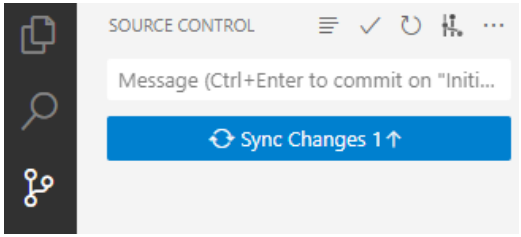
1) *stage* changes



2) Commit-message eingeben
3) Commiten



4) Sync (push↑ und/oder pull ↓)

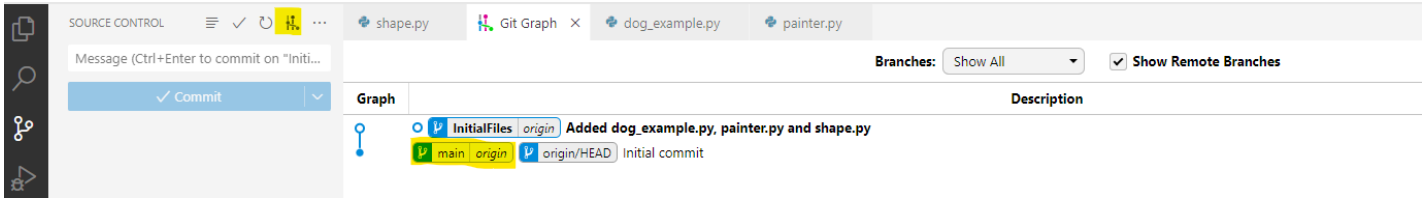


Merge

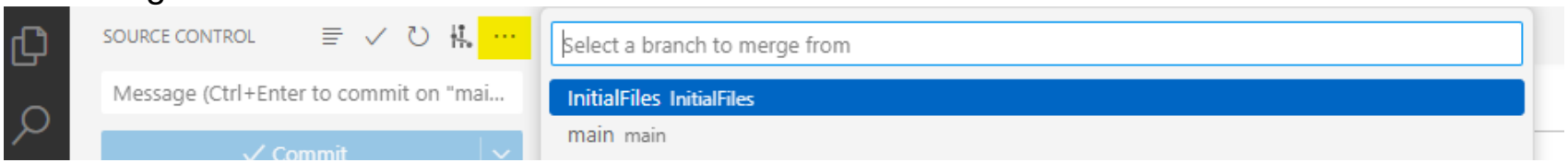
Änderungen von einem Branch zum anderen Branch kopieren & integrieren

- Während der Feature-Entwicklung: Um die anderen Features zu erhalten
  Main Branch → Feature Branch
- Am Ende der Feature-Entwicklung: Um das Feature auf den Master zu schieben
  Feature Branch → Main Branch (Master)

- Wechsle zum Ziel-Branch (derjenige, der die Änderungen erhalten soll)



- Über … - Branch – Merge branch… wähle den Quell-Branch, von dem die Änderungen kommen sollen



- Sync changes (**push** ↑ & pull ↓)