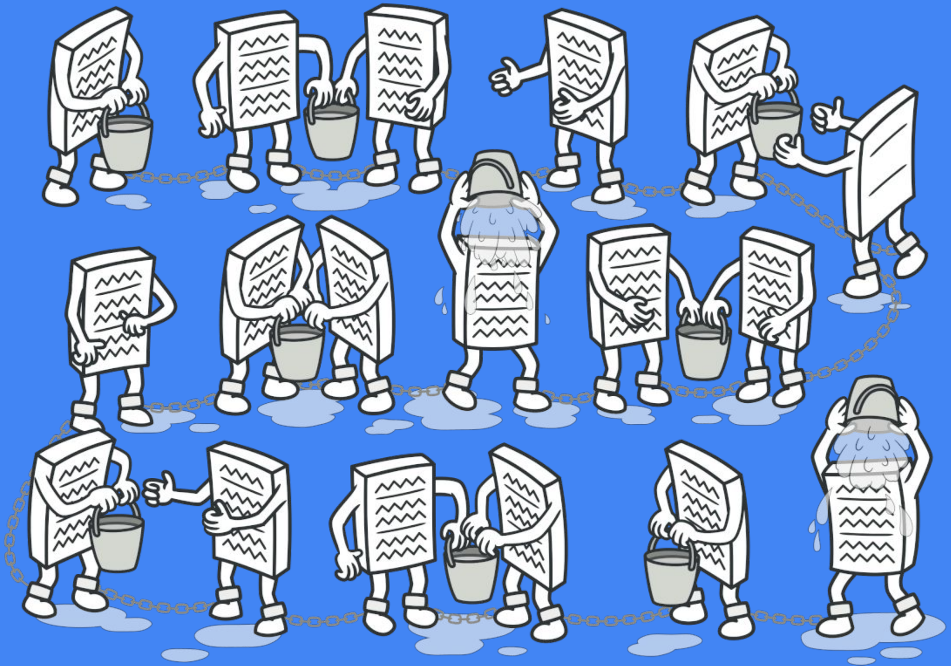
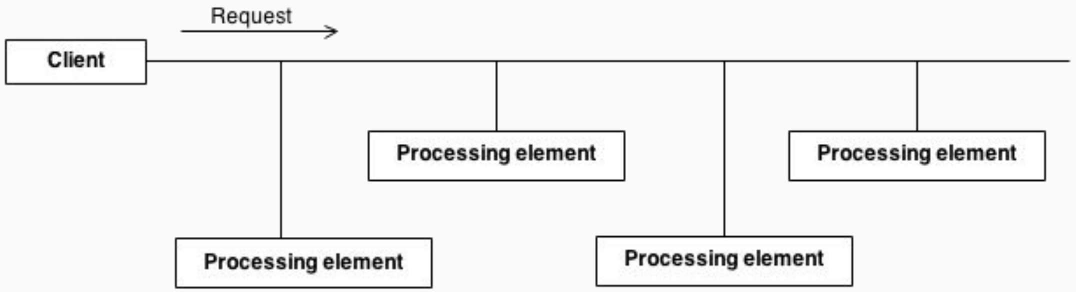
Chain of Responsibility



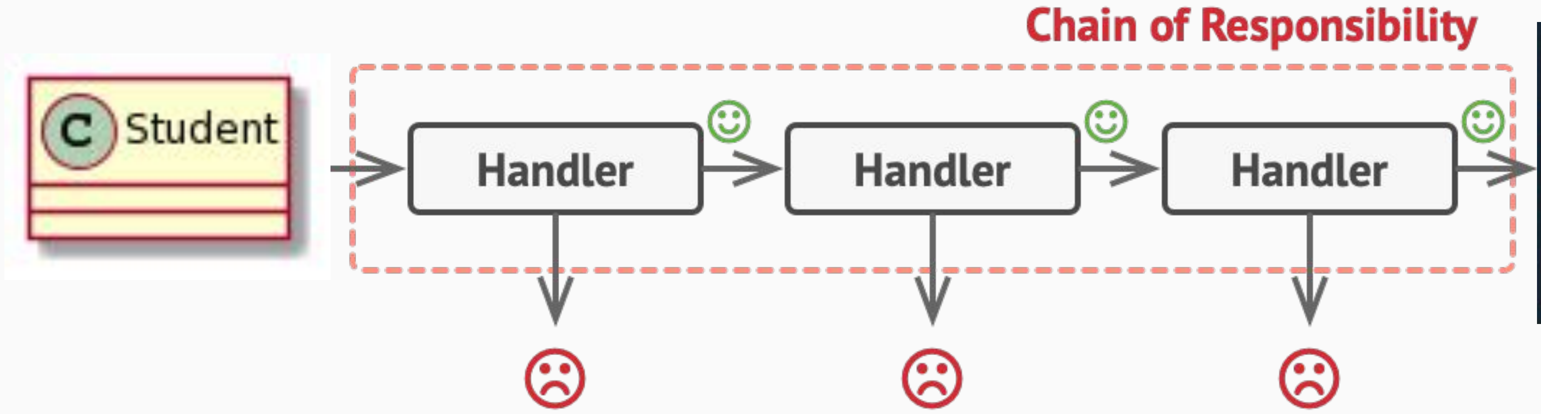
Rinat Mullahmetov

Chain of Responsibility

- Behavioral design pattern that lets you pass requests along a chain of handlers.
- Upon receiving a request, each handler decides either to process the request or to pass it to the next handler in the chain.



Real World Example (Auto P/F Checker)



Auto P/F Checker github

```
@dataclass
class Student:
  name: str
  mail: str

  tp: List[Task]
  td: List[Task]

  final project: Task
  final exam: Task

  attendance: List[bool]
```

```
class Abstract Checker(Checker):
   next checker: Checker = None

  def set_next(self, checker: Checker) ->
Checker:
      self. next checker = checker
      return checker

  @abstractmethod
  def check(self, student: Student) ->
Union[Student, Fail]:
      if self._next_checker:
         return self. next checker.check(student)

      return Fail(None)
```

Students

```
name: Rinat
mail: r.yes@inno.ru
tp:
  - grade: 30
    cheating: True
  - grade: 30
  - grade: 70
  - grade: 30
  - grade: 50
td:
  - grade: 80
  - grade: 70
  - grade: 100
    cheating: True
  - grade: 90
  - grade: 70
final project:
  grade: 100
final exam:
  grade: 100
  cheating: True
attendance: [True, False, True, False, True, True]
```

```
name: Yarick
mail: y.heh@inno.ru
tp:
  - grade: 80
  - grade: 70
  - grade: 100
  - grade: 90
  - grade: 70
td:
  - grade: 80
  - grade: 70
  - grade: 100
  - grade: 90
  - grade: 70
final project:
  grade: 100
 final exam:
  grade: 100
 attendance: [True, False, True, False, True, True]
```

Let’s check!

```
class Attendance Checker(Abstract Checker):
  def __init__(self, thresh: int = 5):
     super().__init__()
     self.thresh = thresh

  def check(self, student: Student):
     if
self.is_attendance_enough(student.attendance):
        return super().check(student)
     else:
        return Fail(self)

  def is_attendance_enough(self, attendance:
List[bool]) -> bool:
     return True if sum(attendance) > self.thresh
else False
```

```
class Tp Checker(Abstract Checker):
  def __init__(self, thresh: int = 56):
     super().__init__()
     self.thresh = thresh

  def check(self, student: Student):
     if self.is_tp_pass(student.tp):
        return super().check(student)
     else:
        return Fail(self)

  def is_tp_pass(self, tp: List[Task]) -> bool:
     grades = [task.grade for task in tp]
     mean = sum(grades) / len(grades)
     return True if mean > self.thresh else False
```

Let’s check

avg_tp_grader.yml

```
- class: Attendance Checker
 params:
   thresh: 2

- class: Tp Checker
 params:
   thresh: 56
```

```
python grade.py
--students  tests/templates/students.yml
--grader tests/templates/avg_tp_grader.yml

Out:

Yarick - Passed
Rinat - Failed by TpChecker
```

Use max TP grade!

```
class Tp Max Checker(Tp Checker):
  def is_tp_pass(self, tp: List[Task]) -> bool:
     grades = [task.grade for task in tp]
     max grade = max(grades)
     return True if max_grade > self.thresh else False
```

Use max TP grade!

max_tp_grader.yml

```
- class: Attendance Checker
  params:
    thresh: 2

- class: Tp Max Checker
  params:
    thresh: 56
```

```
python grade.py
--students  tests/templates/students.yml
--grader tests/templates/max_tp_grader.yml

Out:

Yarick - Passed
Rinat - Passed
```

Cheaters must be punished!

```
class Cheating Checker(Abstract Checker):
  def __init__(self, thresh: int = 2):
     super().__init__()
     self.thresh = thresh

  def check(self, student: Student):
     if self.is_honest(student):
        return super().check(student)
     else:
        return Fail(self)

  def is_honest(self, student: Student) -> bool:
     cheating = []

     for task in student.tp:
        cheating.append(task.cheating)

     for task in student.td:
        cheating.append(task.cheating)

     cheating.append(student.final_exam.cheating)
     cheating.append(student.final_project.cheating)

     return True if sum(cheating) < self.thresh else False
```

Cheaters must be punished!

cheating grader.yml

```
- class: Attendance Checker
  params:
    thresh: 2

- class: Tp Max Checker
  params:
    thresh: 56

- class: Cheating Checker
  params:
    thresh: 2
```

```
python grade.py
--students  tests/templates/students.yml
--grader tests/templates/cheating_grader.yml

Out:

Yarick - Passed
Rinat - Failed by CheatingChecker
```

Still Hungry?

- Take your data from the moodle and check if you pass …
- The final exam was on the last class. Add a check that the student attended the last day.
- Add more tests.

Pros and Cons

+ You can control the order of request handling.
+ Single Responsibility Principle. You can decouple classes that invoke operations from classes that perform operations.
+ Open/Closed Principle. You can introduce new handlers into the app without breaking the existing client code.

- Some requests may end up unhandled.

Questions?

THANK YOU FOR THE ADVENTURE!
COME ALONG WITH ME
PRIMORY.TUMBLR.COM
DON'T REPOST OUT OF TUMBLR