

|  |
| --- |
| **Автономное учреждение  профессионального образования**  **Ханты-Мансийского автономного округа – Югры**  **«СУРГУТСКИЙ ПОЛИТЕХНИЧЕСКИЙ КОЛЛЕДЖ»**  **(АУ «Сургутский политехнический колледж»)** |
|  |
| СТРУКТУРНОЕ ПОДРАЗДЕЛЕНИЕ - 4  (Энергетическое отделение) |
|  |

**Отчет**

**по практической работе**

**«Инструментальные Средства»**

Применение отладочных классов в проекте. Отладка проекта

Выполнил:

студент очной формы обучения

Браславцев Трофим Сергеевич

Сургут 2024

**Введение**

Отладка программного проекта важна для выявления и устранения ошибок в его работе. В данной работе будет изучено применение отладочных классов для упрощения поиска ошибок и отладки модулей.

**Цель работы**

Цель данной работы – изучить и применить отладочные классы в проекте для улучшения процесса поиска и исправления ошибок в коде.

**Практическая часть**

**Задание 1: Создание отладочных классов**

**1. Реализуйте класс для отслеживания выполнения программы.**

import logging

class Debugger:

def \_\_init\_\_(self, log\_file='debug.log'):

self.logger = logging.getLogger('Debugger')

self.logger.setLevel(logging.DEBUG)

formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')

file\_handler = logging.FileHandler(log\_file)

file\_handler.setFormatter(formatter)

self.logger.addHandler(file\_handler)

def log(self, message, level=logging.DEBUG):

self.logger.log(level, message)

def log\_exception(self, e):

self.logger.exception(f"Ошибка: {e}")

**2. Добавьте возможность логирования исключений в классе.**

import logging

class Debugger:

def \_\_init\_\_(self, log\_file='debug.log'):

self.logger = logging.getLogger('Debugger')

self.logger.setLevel(logging.DEBUG)

formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')

file\_handler = logging.FileHandler(log\_file)

file\_handler.setFormatter(formatter)

self.logger.addHandler(file\_handler)

def log(self, message, level=logging.DEBUG):

self.logger.log(level, message)

def log\_exception(self, e):

self.logger.exception(f"Ошибка: {e}")

# Пример использования

debugger = Debugger()

debugger.log("Начало выполнения программы")

try:

result = 10 / 0

except ZeroDivisionError as e:

debugger.log\_exception(e)

**Задание 2: Отладка проекта с использованием отладочных классов**

**1. Внедрите отладочный класс в основной проект.**

# main.py

from debugger import Debugger

debugger = Debugger()

def main():

debugger.log("Начало выполнения программы")

try:

result = 10 / 0

except ZeroDivisionError as e:

debugger.log\_exception(e)

debugger.log("Конец выполнения программы")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**2. Проведите тестирование, используя функционал класса для отслеживания ошибок.**

# test.py

from main import main

main()

**Задание 3: Использование отладочного класса для анализа производительности**

**1. Реализуйте возможность измерения времени выполнения методов в отладочном классе.**

import time

class Debugger:

def \_\_init\_\_(self, log\_file='debug.log'):

self.logger = logging.getLogger('Debugger')

self.logger.setLevel(logging.DEBUG)

formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')

file\_handler = logging.FileHandler(log\_file)

file\_handler.setFormatter(formatter)

self.logger.addHandler(file\_handler)

def log(self, message, level=logging.DEBUG):

self.logger.log(level, message)

def log\_exception(self, e):

self.logger.exception(f"Ошибка: {e}")

def measure\_time(self, func):

def wrapper(\*args, \*\*kwargs):

start\_time = time.time()

result = func(\*args, \*\*kwargs)

end\_time = time.time()

self.log(f"Время выполнения {func.\_\_name\_\_}: {end\_time - start\_time} секунд")

return result

return wrapper

**2. Проанализируйте производительность одного из модулей проекта.**

# main.py

from debugger import Debugger

debugger = Debugger()

@debugger.measure\_time

def slow\_function():

time.sleep(2)

def main():

debugger.log("Начало выполнения программы")

slow\_function()

debugger.log("Конец выполнения программы")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Задание 4: Интеграция отладочного класса с логированием**

**1. Добавьте возможность автоматического сохранения логов в файл для дальнейшего анализа.**

import logging

class Debugger:

def \_\_init\_\_(self, log\_file='debug.log'):

self.logger = logging.getLogger('Debugger')

self.logger.setLevel(logging.DEBUG)

formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')

file\_handler = logging.FileHandler(log\_file)

file\_handler.setFormatter(formatter)

self.logger.addHandler(file\_handler)

def log(self, message, level=logging.DEBUG):

self.logger.log(level, message)

def log\_exception(self, e):

self.logger.exception(f"Ошибка: {e}")

def measure\_time(self, func):

def wrapper(\*args, \*\*kwargs):

start\_time = time.time()

result = func(\*args, \*\*kwargs)

end\_time = time.time()

self.log(f"Время выполнения {func.\_\_name\_\_}: {end\_time - start\_time} секунд")

return result

return wrapper

**2. Проверьте корректность логирования в проекте.**

# main.py

from debugger import Debugger

debugger = Debugger()

@debugger.measure\_time

def slow\_function():

time.sleep(2)

def main():

debugger.log("Начало выполнения программы")

slow\_function()

try:

result = 10 / 0

except ZeroDivisionError as e:

debugger.log\_exception(e)

debugger.log("Конец выполнения программы")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Задание 5: Отладка многопоточной программы с помощью отладочного класса**

**1. Примените отладочный класс для отслеживания ошибок в многопоточной программе.**

import threading

from debugger import Debugger

debugger = Debugger()

def worker():

debugger.log("Начало работы потока")

try:

result = 10 / 0

except ZeroDivisionError as e:

debugger.log\_exception(e)

debugger.log("Конец работы потока")

threads = []

for i in range(5):

t = threading.Thread(target=worker)

threads.append(t)

t.start()

for t in threads:

t.join()

debugger.log("Все потоки завершили работу")

**2. Проверьте, корректно ли отслеживаются ошибки в разных потоках.**

import threading

from debugger import Debugger

debugger = Debugger()

def worker():

debugger.log("Начало работы потока")

try:

result = 10 / 0

except ZeroDivisionError as e:

debugger.log\_exception(e)

debugger.log("Конец работы потока")

threads = []

for i in range(5):

t = threading.Thread(target=worker)

threads.append(t)

t.start()

for t in threads:

t.join()

debugger.log("Все потоки завершили работу")

**Вывод**

В данной работе были изучены и применены отладочные классы для улучшения процесса поиска и исправления ошибок в коде. Были созданы отладочные классы, внедрены в проект, использованы для анализа производительности, интегрированы с логированием и применены для отладки многопоточной программы. Правильное использование отладочных классов значительно упрощает процесс отладки и повышает качество программного обеспечения.