Цель: закрепить навыки объектно-ориентированного программирования на языке Python

Ход работы

Задание 1 (Вариант 5): Реализовать простой класс. Множество целых чисел ограниченной мощности – Предусмотреть возможность объединения двух множеств, вывода на консоль элементов множества, а также метод, определяющий, принадлежит ли указанное значение множеству. Класс должен содержать методы, позволяющие добавлять и удалять элемент в/из множества. Конструктор должен позволять создавать объекты с начальной инициализацией. Мощность множества задается при создании объекта. Реализацию множества осуществить на базе списка. Переопределить метод __eq__, выполняющий сравнение объектов данного типа.

Код программы:

```
class LimitedSet:
   def __init__(self, capacity, items=None):
       if capacity <= 0: raise ValueError("Мощность > 0")
       self._cap, self._items = capacity, []
       if items:
           unique = []
           for x in items:
               if isinstance(x, int) and x not in unique: unique.append(x)
           if len(unique) > capacity: raise ValueError("Превышение мощности")
           self._items = unique

   @property
   def cap(self): return self._cap
   @property
   def size(self): return len(self._items)

   def add(self, x):
       if not isinstance(x, int): raise TypeError("Нужно целое")
       if x in self._items: return False
       if len(self._items) >= self._cap: raise OverflowError("Множество полно")
       self._items.append(x); return True

   def remove(self, x):
       if isinstance(x, int) and x in self._items:
           self._items.remove(x); return True
       return False

   def contains(self, x): return isinstance(x, int) and x in self._items

   def union(self, other):
```

```
        if not isinstance(other, LimitedSet): raise TypeError("Ожидается
LimitedSet")
        unique = list(set(self._items + other._items))
        if len(unique) > self._cap + other._cap: raise ValueError("Не хватит места")
        return LimitedSet(self._cap + other._cap, unique)

    def display(self):
        print(f"Элементы: {', '.join(map(str, self._items)) if self._items else
'пусто'} [{self.size}/{self._cap}]")

    def __str__(self):
        return f"LimitedSet({{{', '.join(map(str, self._items))}}},
{self.size}/{self._cap})" if self._items else f"LimitedSet(пусто, {self._cap})"

    def __eq__(self, other):
        return isinstance(other, LimitedSet) and set(self._items) ==
set(other._items)


# Пример
if __name__ == "__main__":
    s1 = LimitedSet(5, [1,2,3])
    s2 = LimitedSet(3); s2.add(10); s2.add(20)
    print(s1); print(s2); print(f"Содержит 2? {s1.contains(2)}")
    s1.remove(2); print(s1)
    s3 = s1.union(s2); print(f"Объединение: {s3}")
    print(f"Равны? {s1 == LimitedSet(5, [1,3])}")
```

Результат работы программы:

```
/Users/svyat/PycharmProjects/PythonProject/Lab2SPP/.ve
LimitedSet({1, 2, 3}, 3/5)
LimitedSet({10, 20}, 2/3)
Содержит 2? True
LimitedSet({1, 3}, 2/5)
Объединение: LimitedSet({1, 10, 3, 20}, 4/8)
Равны? True
```

Рисунок 1. Результат выполнения программы

Задание 2 (Вариант 5): Построить модель программной системы с применением отношений (обобщения, агрегации, ассоциации, реализации) между классами. Задать атрибуты и методы классов. Реализовать (если необходимо) дополнительные классы. Продемонстрировать работу разработанной системы.Система Библиотека. Читатель оформляет Заказ на Книгу. Система осуществляет поиск в Каталоге. Библиотекарь выдает Читателю Книгу на абонемент или в читальный зал. При невозвращении

Книги Читателем он может быть занесен Администратором в «черный список».

Код программы:

```
from abc import ABC, abstractmethod
from enum import Enum


class BookStatus(Enum): AVAILABLE = 1; ISSUED = 2


class BookType(Enum): ABONEMENT = 1; READING_ROOM = 2


class Entity(ABC):
   def __init__(self, id): self.id = id


class Searchable(ABC):
   @abstractmethod
   def search(self, q): pass


class Book(Entity):
   def __init__(self, id, t, a): super().__init__(
       id); self.title = t; self.author = a; self.status = BookStatus.AVAILABLE;
self.type = None


class Reader(Entity):
   def __init__(self, id, n):
       super().__init__(id); self.name = n; self.blacklisted = False; self.books =
[]

   def borrow(self, b):
       if self.blacklisted: raise Exception("В ЧС")
       if b.status != BookStatus.AVAILABLE: raise Exception("Недоступна")
       self.books.append(b);
       b.status = BookStatus.ISSUED

   def ret(self, b):
       if b in self.books: self.books.remove(b); b.status = BookStatus.AVAILABLE;
return True
       return False


class Librarian(Entity):
   def __init__(self, id, n): super().__init__(id); self.name = n

   def issue(self, r, b, t):
       if b.status != BookStatus.AVAILABLE: raise Exception("Недоступна")
       b.type = t;
       r.borrow(b);
       return f"{self.name} выдал {b.title} {r.name}"
```

```
class Admin(Entity):
   def __init__(self, id, n): super().__init__(id); self.name = n

   def bl_add(self, r): r.blacklisted = True

   def bl_rem(self, r): r.blacklisted = False


class Order:
   def __init__(self, id, r, b): self.id = id; self.reader = r; self.book = b;
self.done = False


class Catalog(Searchable):
   def __init__(self): self.books = []

   def add(self, b): self.books.append(b)

   def search(self, q): return [b for b in self.books if q.lower() in
b.title.lower() or q.lower() in b.author.lower()]

   def get(self, id): return next((b for b in self.books if b.id == id), None)


class Library:
   def __init__(self):
       self.cat = Catalog(); self.readers = []; self.libs = []; self.admins = [];
self.orders = []

   def add_r(self, r):
       self.readers.append(r)

   def add_l(self, l):
       self.libs.append(l)

   def add_a(self, a):
       self.admins.append(a)

   def ord(self, r, b):
       if r.blacklisted: raise Exception("В ЧС")
       if b.status != BookStatus.AVAILABLE: raise Exception("Недоступна")
       o = Order(len(self.orders) + 1, r, b);
       self.orders.append(o);
       return o

   def proc(self, o, l, t):
       if o.done: raise Exception("Уже выполнен")
       r = l.issue(o.reader, o.book, t);
       o.done = True;
       return r


if __name__ == "__main__":
   lib = Library()
   b1 = Book(1, "Война и мир", "Толстой");
   b2 = Book(2, "Преступление", "Достоевский");
   b3 = Book(3, "Мастер", "Булгаков")
   for b in [b1, b2, b3]: lib.cat.add(b)
```

```
lib.add_l(Librarian(1, "Анна"));
lib.add_a(Admin(1, "Иван"))
r1 = Reader(1, "Петр");
r2 = Reader(2, "Мария");
lib.add_r(r1);
lib.add_r(r2)

print("Поиск:", [b.title for b in lib.cat.search("пре")])
o = lib.ord(r1, lib.cat.get(1));
print(f"Заказ: {o.id}")
print(lib.proc(o, lib.libs[0], BookType.ABONEMENT))
print(f"Книги Петра: {[b.title for b in r1.books]}")

try:
    lib.ord(r2, b1)
except Exception as e:
    print(f"Ошибка: {e}")

r1.ret(b1);
print("Книга возвращена")
lib.admins[0].bl_add(r2);
print(f"Мария в ЧС: {r2.blacklisted}")
try:
    lib.ord(r2, b2)
except Exception as e:
    print(f"Ошибка: {e}")

lib.admins[0].bl_rem(r2);
o2 = lib.ord(r2, lib.cat.get(3))
print(lib.proc(o2, lib.libs[0], BookType.READING_ROOM))
print(f"Всего заказов: {len(lib.orders)}")
```

Результат работы программы:

```
Книги Петра: ['Война и мир']
Ошибка: Недоступна
Книга возвращена
Мария в ЧС: True
Ошибка: В ЧС
Анна выдал Мастер Мария
Всего заказов: 2
```

Рисунок 2. Результат выполнения программы

Вывод: закрепил навыки объектно-ориентированного программирования на языке Python