Объектно-ориентированный дизайн

Обзор урока

- Композиция и наследование
- Инкапсуляция (уровни доступа)
- Полиморфизм (связывание)
- Абстактный класс и интерфейс
- Внутренний и анонимный класс
- Инверсия зависимостей
- ООП дизайн
- Паттерн стратегия
- Обработка коллекций

Материалы для самостоятельного изучения

- Брюс Эккель, Философия Java (главы 7-10)
- Джошуа Блох, Эффективная Java

Домашнее задание

Алгоритмы. Изучить структуру данных бинарная куча и реализовать на ее основе очередь с приоритетами. Реализовать всего 2 операции — добавление нового элемента по ключу и извлечение (с удалением) элемента с максимальным приоритетом

```
// K — ключ (приоритет), приоритеты можно
   сравнивать
// T — значение (некий объект, который
   доступен по ключу)

interface PriorityQueue<K extends
   Comparable<K>, T> {
```

```
    T getMax();
    void insert(K k, T t);
}
```

Для обработки коллекций удобно воспользоваться подходом из функционального программирования. Основными операциями на коллекция являются фильтрация, свертка и преобразование. На уроке были рассмотрены операции filter (фильтрация) и reduce (свертка). Нужно помнить, что начальная коллекция не должна модифицироваться, все изменения делаются в копии. Паттерн проектирования Стратегия используется для реализации конкретной операции или фильтра. Например, если применить фильтрацию с условием (>=10) на списке (1, 4, 8, 34, 9, 12), то на выходе получим отфильтрованный список (34,12). Пример свертки - это сумма всех элементов списка (1,2,3,4,5) -> 15.

```
    // Принимает элемент T и проверяет его на
       условие
interface Predicate<T> {
    boolean test(T t);
}

// Принимает 2 аргумента одного типа, и делает
   над ними операцию
interface BiOperator<T> {
    T apply(T t, T tt);
}
/*
 * Проверить элементы коллекции на заданное
    условие.
 * Вернуть коллекцию элементов, прошедших фильтр
 */
static <T> List<T> filter(List<T> list,
    Predicate<T> predicate) {
    List<T> result = new ArrayList<>();
    for (T t : list) {
        if (predicate.test(t)) {
            result.add(t);
        }
    }
    return result;
```

```
}

/*
 * Последовательно применить операцию ко всем
    элементам коллекции
 * Вернуть одно значение
 */
static <T> T reduce(List<T> list, T init,
   BiOperator<T> op) {
   for (T t : list) {
      init = op.apply(init, t);
   }
   return init;
}
```

Задание.

Написать свою реализацию map. Эта операция делает преобразование каждого элемента коллекции. На вход подается коллекция объектов типа T и функциональный интерфейс с одной операцией, которая делает преобразование каждого элемента коллекции T -> R. Вернуть из функции преобразованную коллекцию типа R, начальную коллекцию не модифицировать. Пример использования:

```
   interface Operator<R, T> {
      R apply(T t);
   }

   // В этом примере типы R и T совпадают ==
      Integer
   List<Integer> numbers = Arrays.asList(1, 2, 3);
   List<Integer> res = map(numbers, new
      SquareOperator()); // SquareOperator()
      реализует интерфейс Operator
   // 1, 4, 9 - на выходе коллекция квадратов
      чисел

   List<Person> persons = ... // инициализируем
      коллекцию Person
   // У класса Person есть поле age - возраст.
```

```
        // Преобразовываем коллекцию Person -> Integer
           (коллекция возрастов)
        List<Integer> ages = map(persons, new
           AgeOperator());
```