Министерство высшего образования и науки РФ

Санкт–Петербургский политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа компьютерных технологий и информационных систем

**Отчет**

по дисциплине «Системный подход в разработке ПО»

**Обобщенные классы**

**Выполнил:**

студент группы 5130902/20201 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. И. Сафонов

подпись

**Проверил:**

доцент, к.т.н \_\_\_\_\_\_\_\_\_\_\_\_ С. А. Нестеров

подпись

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_ 2025г.

Санкт-Петербург

2025

Оглавление

[2. Лекция 2.3 3](#_Toc190780471)

[2.1. Задание 1 3](#_Toc190780472)

[2.2. Пример instanceof 4](#_Toc190780473)

[2.3. Задание 2 6](#_Toc190780474)

[2.4. Пример на кастинг 8](#_Toc190780475)

[3. Практика 2.3 9](#_Toc190780476)

[4. Лекция 2.4 14](#_Toc190780477)

[4.1. Обработка ошибок 14](#_Toc190780478)

# Васильев

## Теория

В обобщенных классах тип данных выступает параметром. Обычно обобщенные классы используют в тех случаях, когда с помощью класса реализуется некоторая «структура» или «конструкция», которая имеет универсальные характеристики и слабо зависит от конкретного типа данных, с которыми приходится оперировать. Это же замечание относится и к методам. В таких случаях удобно использовать обобщенные методы, в которых тип данных является параметром. Далее обсуждаются методы создания и использования обобщенных классов и методов, а также ряд смежных вопросов.

При описании обобщенного класса некий формальный параметр (или параметры) декларируются как такие, что обозначают тип данных, а при создании объекта класса идентификатор типа передается как параметр. Параметры, которые отождествляются в описании обобщенного класса с типом данных, указываются в угловых скобках в описании класса после имени класса. Параметр (или параметры) типа просто перечисляются через запятую, и эти параметры могут использоваться в описании обобщенного класса. В качестве параметра типа может использоваться в принципе любой идентификатор — главное, чтобы это не было одно из зарезервированных ключевых слов.

Также идентификаторы указываются в угловых скобках в инструкции создания объекта — между названием класса и круглыми скобками с аргументами конструктора. Также приемлемо использовать упрощенный синтаксис: идентификаторы типов во вторых угловых скобках (в команде вызова конструктора) можно не указывать (но при этом угловые скобки должны присутствовать).

Помимо обобщенных классов, в Java можно создавать обобщенные методы. В описании обобщенного метода тип определяется через обобщенный параметр (или параметры), а при вызове метода значения обобщенных параметров определяются на основе типов аргументов, переданных методу. Обычно (но не обязательно) обобщенные методы являются статическими.

При описании обобщенного метода обобщенный параметр типа в угловых скобках указывается перед идентификатором типа результата метода. При вызове метода это дает возможность идентифицировать значения обобщенных параметров, которые следует использовать при выполнении кода метода.

Нестатический обобщенный метод создается практически точно так же, как и статический. Принципиальное отличие в том, что нестатический метод вызывается из объекта.

Конкретные реализации обобщенного суперкласса могут использоваться в качестве суперкласса при наследовании. Проще говоря, при создании подкласса в качестве суперкласса можно указать обобщенный класс — но не как таковой, а с конкретными значениями для обобщенных параметров.

При использовании обобщенных классов и методов в инструкции объявления обобщенного типа может использоваться выражение вида X extends суперкласс, означающее, что обобщенный тип X должен быть таким, что является подклассом (прямым или опосредованным) класса суперкласс. В некоторых случаях такой подход бывает достаточно удобен.

Обобщенным может быть не только класс или метод, но и интерфейс. Объявляется обобщенный интерфейс аналогично к обобщенному классу: после имени интерфейса в угловых скобках указываются названия для обобщенных типов. Такой обобщенный интерфейс может использоваться как для создания обобщенного класса, так и для создания класса с конкретными значениями для обобщенных параметров типа.

Использование параметра обобщенного типа в классе или методе подразумевает, что при создании объекта класса или вызове метода происходит определение (идентификация) обобщенного типа. Но нередко значение для обобщенного параметра играет формальную роль. В таких случаях удобно использовать обобщенные подстановки.

Идея обобщенной подстановки в принципе проста: используя в качестве определения обобщенного параметра инструкцию вида можно сообщить компилятору, что в соответствующем месте используется некоторый неизвестный обобщенный тип.

При использовании обобщенных подстановок можно использовать огра ничения: с помощью ключевого слова extends определяют подстановки, обозначающие подклассы для данного класса, а с помощью ключевого слова super определяют подстановки для суперклассов данного класса. В частности, выражение вида означает, что для параметра обобщенного типа используется неустановленный тип, являющийся подклассом (прямым или опосредованным) класса SuperClass. Выражение вида означает, что в качестве параметра обобщенного типа может быть класс, являющийся (прямо или через цепочку наследования) суперклассом для класса SubClass.

## Запуск примеров

Использование обобщенного класса

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Несколько обобщенных параметров

Изображение выглядит как текст, снимок экрана, программное обеспечение, компьютер

Контент, сгенерированный ИИ, может содержать ошибки.

Использование статических обобщенных методов

Изображение выглядит как текст, снимок экрана, программное обеспечение, компьютер

Контент, сгенерированный ИИ, может содержать ошибки.

Использование обобщенных методов

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Наследование обобщенных классов

Изображение выглядит как текст, снимок экрана, программное обеспечение, компьютер

Контент, сгенерированный ИИ, может содержать ошибки.

Использование обобщенных методов с параметром, который ограничен по наследованию

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Использование обобщенного интерфейса

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Больше обобщенных интерфейсов

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Wildcard

Изображение выглядит как текст, снимок экрана, программное обеспечение, компьютер

Контент, сгенерированный ИИ, может содержать ошибки.

BoundedWildcard

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

# Блинов, Романчик

При разработке приложений достаточно часто встречаются сущности, число значений которых ограничено естественным образом: страны мира, месяцы года, дни недели, марки транспортных средств, типы пользователей и многие другие.

Типобезопасные перечисления (typesafe enums) в Java представляют собой классы и являются подклассами абстрактного класса java.lang.Enum. Вместо слова class при описании перечисления используется слово enum. При этом объекты перечисления инициализируются прямым объявлением без помощи оператора new. При инициализации хотя бы одного перечисления происходит инициализация всех без исключения оставшихся элементов данного перечисления.

В операторах case используются константы без уточнения типа перечисления, так как его тип определен в switch. Перечисление как подкласс класса Enum может содержать поля, конструкторы и методы, реализовывать интерфейсы. Каждый элемент enum может использовать методы:

static enumType[] values() — возвращает массив, содержащий все элементы перечисления в порядке их объявления;

static > T valueOf(Class enumType, String arg) — создает элемент перечисления, соответствующий заданному типу и значению передаваемой строки;

static enumType valueOf(String arg) — создает элемент перечисления, соответствующий значению передаваемой строки;

int ordinal() — возвращает позицию элемента перечисления, начиная с нуля, следствием чего является возможность сравнения элементов перечисления между собой на больше\меньше соответствующими операторами;

String name() — возвращает имя элемента, так же как и toString();

int compareTo(T t) — сравнивает элементы на больше, меньше либо равно.

Класс перечисления может объявлять собственные методы, и, следовательно, экземпляры перечисления могут к этим методам обращаться. Перечисления представляют собой классы, а классам положено явно или неявно объявлять конструкторы

Перечисление является классом, поэтому в его теле можно объявлять, кроме методов, также поля и конструкторы. Все конструкторы вызываются автоматически при инициализации любого из элементов. Конструктор не может быть объявлен со спецификаторами public и protected, так как не вызывается явно и перечисление не может быть суперклассом. Поля перечисления используются для сохранения дополнительной информации, связанной с его элементом. Метод toString() реализован в классе Enum для вывода элемента в виде строки. Если переопределить метод toString() в конкретной реализации перечисления, то можно выводить не только значение элемента, но и значения его полей, то есть предоставить полную информацию об объекте, как и определяется контрактом метода. Метод ordinal() определяет порядковый номер элемента перечисления, обратная задача не совсем тривиальна, но решаема. Определение элемента перечисления по позиции с применением метода values() преобразования элементов перечисления в массив

На перечисления накладывается целый ряд ограничений.

Им запрещено:

• быть суперклассами;

• быть подклассами;

• быть абстрактными;

• быть параметризированными;

• создавать экземпляры, используя ключевое слово new.

Ключевое слово record представляет еще один способ создать класс с неизменяемыми экземплярами. Этот класс является подклассом класса java.lang.Record и, естественно, не может наследовать другой класс, а также сам не может выступать в роли суперкласса. Реализация интерфейсов разрешается. Также класс record может объявлять собственные методы.

До создания перечислений

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

После создания перечислений

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Площадь фигур

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Запись

**Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.**

# Задания к лекции

Добавление перечисления в JavaBank

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Итерация по перечислению

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Контент, сгенерированный ИИ, может содержать ошибки.

Добавление второго поля

Изображение выглядит как текст, электроника, снимок экрана, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Обобщенный класс

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, электроника, снимок экрана, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Дженерики

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Wildcards без ограничений

Изображение выглядит как текст, Шрифт, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Wildcards с ограничениями (upper)

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Wildcards (lower)

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

# Практика

Лексика

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Контент, сгенерированный ИИ, может содержать ошибки.

1. Collection
2. Generic
3. Enumeration

Добавление перечисления

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Добавление возможности выбора (даже если будет кредит, то добавится savings)

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Изменение проекта с байками

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Кубоид

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, электроника, снимок экрана, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

# Алгоритмы

Чтобы понять работу алгоритма, недостаточно просто рассмотреть его шаги. Нужно выяснить еще несколько важных моментов.

* Поведение алгоритма. Находит ли он наилучшее из возможных или просто хорошее решение? Может ли быть несколько наилучших решений? Есть ли смысл отдать предпочтение одному из них?
* Скорость алгоритма. Быстрый он или нет? Замедляет ли работу только с некоторыми входными данными?
* Требования к памяти алгоритма. Сколько компьютерных ресурсов ему необходимо? Является ли такой объем приемлемым? Нужны ли алгоритму миллионы терабайтов памяти, которыми компьютер не располагает (по крайней мере, сейчас)?
* Основные методы, используемые в алгоритме. Можно ли задействовать их повторно для решения подобных задач?

Хороший алгоритм должен быть правильным, надежным и эффективным. Если он не справляется с поставленной задачей, от него мало пользы. Нет смысла в нем и тогда, когда он дает неправильное решение.

Если алгоритм не является надежным, использовать его в программе становится небезопасно. По-настоящему хороший код должен быть простым, интуитивно понятным и изящным. Только в этом случае вы получите правильный результат или сможете внести необходимые изменения в структуру программы. Если алгоритм сложный и запутанный, у вас возникнут проблемы не только с его реализацией, но и с устранением возможных ошибок. Под сомнение можно будет поставить и сам результат: ведь если алгоритм непонятен, как узнать, правильно он работает или нет?

Большинство разработчиков много времени уделяют эффективности алгоритмов. И не зря. Правильный результат, легкость реализации и возможность исправить ошибки — все это сведется к нулю, если алгоритм окончит свою работу только через семь лет или потребует объем памяти больше того, что есть в компьютере.

Асимптотическая сложность (производительность) определяется функцией, которая указывает, насколько ухудшается работа алгоритма с усложнением поставленной задачи. Такую функцию записывают в круглых скобках, предваряя прописной буквой О.

ЗАМЕЧАНИЕ. Выражение O(N2) читается как «порядок N в квадрате». Существуют пять основных правил для расчета асимптотической сложности алгоритма.

1. Если для математической функции f алгоритму необходимо выполнить определенные действия f(N) раз, то для этого ему понадобится сделать O(f(N)) шагов.

2. Если алгоритм выполняет одну операцию, состоящую из O(f(N)) шагов, а затем вторую операцию, включающую O(g(N)) шагов, то общая производительность алгоритма для функций f и g составит O(f(N) + g(N)).

3. Если алгоритму необходимо сделать O(f(N) + g(N)) шагов и область значений N функции f(N) больше, чем у g(N), то асимптотическую сложность можно упростить до выражения O(f(N)).

4. Если алгоритму внутри каждого шага O(f(N)) одной операции приходится выполнять еще O(g(N)) шагов другой операции, то общая производительность алгоритма составит O(f(N) g(N)).

5. Постоянными множителями (константами) можно пренебречь. Если C является константой, то O(C f(N)) или O(f(C N)) можно записать как O(f(N)). Приведенные правила кажутся немного формальными из-за абстрактных функций f(N) и g(N), но на самом деле ими очень легко пользоваться на практике. Ниже приведено несколько примеров, которые облегчат понимание.

**sqrt (N)** Алгоритмы с производительностью O(sqrt (N)), где sqrt — функция извлечения квадратного корня, не являются общими и не рассматриваются в данной книге. Эта функция возрастает очень медленно, хотя и несколько быстрее, чем log (N).

**N** Алгоритм FindLargest (см. пункт «Правило 1» подраздела «Асимптотическая сложность алгоритма» в разделе «Свойство алгоритма» текущей главы) имеет производительность O(N). Функция N возрастает быстрее, чем log (N) и sqrt (N), но все же не так быстро, поэтому большинство подобных алгоритмов демонстрирует на практике хорошую производительность.

**N log N** Предположим, что алгоритм перебирает все элементы в поставленной задаче, а затем в отдельном цикле выполняет с элементом какую-то операцию O(log (N)). В этом случае производительность алгоритма определяется выражением O(N log N) или O(N log N). Как вариант, задача может выглядеть так: алгоритм выполняет операцию O(log (N)) и на каждом ее шаге что-то делает с каждым элементом.

Предположим, у вас есть отсортированное дерево, содержащее N элементов (как описано выше) и такой же размерности массив. Вам надо узнать, какие из элементов массива присутствуют в дереве. Один из способов выяснить это — осуществить циклическое прохождение по величинам массива, задействовав описанный ранее метод поиска по дереву. В процессе работы алгоритм проверит N элементов и выполнит log (N) шагов для каждого из них, так что общая производительность будет O(N log N).

Для многих алгоритмов сортировки, основанных на сравнении элементов, время работы составляет O(N log N). На самом деле можно доказать, что любой подобный алгоритм проходит как минимум такое количество шагов, что очень удобно для определения асимптотической сложности. Однако некоторые алгоритмы являются все же более быстрыми благодаря целым числам, которые игнорируются асимптотическим обозначением.

N2 Алгоритм, который изначально перебирает все входные данные, а затем пересматривает их еще раз для каждого единичного значения, имеет производительность O(N2 ). Например, алгоритм ContainsDuplicates (см. пункт «Правило 4» подраздела «Асимптотическая сложность алгоритма» в разделе «Свойство алгоритма» текущей главы) выполняется именно за такое время. Возможны и другие степени N, предположим O(N3 ) или O(N4 ). Однако они являются значительно более медленными, чем O(N2 ).

Говорят, алгоритм имеет многочленное время работы, если оно включает в себя многочлен N: например O(N), O(N2 ), O(N6 ) и даже O(N4000). В любом случае подобные задачи можно решить. А вот экспоненциальное и факториальное время работы, описанное ниже, возрастает очень быстро, поэтому алгоритмы с указанной производительностью будут применимы только для небольшого количества входных данных.

2N Экспоненциальные функции, такие как 2N, возрастают молниеносно и поэтому полезны для решения ограниченного круга задач. Обычно посредством алгоритмов с подобным временем работы ищется оптимальный набор входных данных.

В качестве примера рассмотрим следующую задачу. Допустим, у вас есть рюкзак определенной грузоподъемности и набор принадлежностей, каждый из которых имеет свою массу и значение. Вам необходимо наполнить рюкзак предметами так, чтобы их общее значение оказалось наибольшим.

Задача может показаться простой, но все известные алгоритмы для нахождения наилучшего решения потребуют изучить каждую возможную комбинацию предметов. Если учесть, что любой предмет способен пребывать только в двух состояниях (находиться в рюкзаке или вне его), то, умножив количество состояний на количество предметов, получим 2 2 ... 2 = 2N вариантов выбора.

N! Функция N! (читается как «N факториал») рекомендуется для работы с целыми числами больше 0 и определяется формулой N! = 1 2 3 ... N. Она возрастает намного быстрее, чем экспоненциальная функция 2N. В алгоритмах с факториальным временем работы, как правило, ищется оптимальное распределение входных данных. Например, у торгового представителя имеется список городов. Его задача — составить маршрут таким образом, чтобы посетить каждый населенный пункт один раз и вернуться в отправную точку, преодолев минимальное расстояние. Если городов немного, то все просто, но если список длинный, задача усложняется на порядок. Наиболее очевидный способ решения — перебрать все возможные варианты маршрута. Так, с помощью алгоритма можно выбрать N близлежащих населенных пунктов для первого города. После этого у вас останется N – 1 возможных пунктов, которые еще нужно посетить, затем N – 2 и т. д. Следовательно, общее количество вариантов маршрутов составит N (N – 1) (N – 2) ... 1 = N!

Упражнения

1. O(n)
2. Ответ

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

1. Первый при n больше 50, второй при меньше либо равно 50
2. N < 4,92 и N > 15,77
3. O(N2)
4. O(N2)
5. O(N2)
6. O(N)
7. O(N)
8. 1 – 1, 2 – 4, 3 – 10, 4 – 20, 5 – 35, 6 – 56, 7 – 84, 8 - 120