Содержание

3 Специализация: ООП и исключения **1**
3.1 В предыдущем разделе 1
3.2 В этом разделе 1
3.3 Перечисления 1

3. Специализация: ООП и исключения

3.1. В предыдущем разделе

Была рассмотрена реализация объектно-ориентированного программирования в Java. Рассмотрели классы и объекты, а также наследование, полиморфизм и инкапсуляцию. Дополнительно был освещён вопрос устройства памяти.

3.2. В этом разделе

В дополнение к предыдущему, будут разобраны такие понятия, как внутренние и вложенные классы; процессы создания, использования и расширения перечислений. Более детально будет разобрано понятие исключений и их тесная связь с многопоточностью в Java. Будут рассмотрены исключения с точки зрения ООП, процесс обработки исключений.

— Внутренний класс;
— Вложенный класс;
— Исключение;
— Многопоточность;

3.3. Перечисления

Кроме восьми примитивных типов данных и классов в Java есть специальный тип, выведенный на уровень синтаксиса языка - `enum` или перечисление. Перечисления представляют набор логически связанных констант. Объявление перечисления происходит с помощью оператора `enum`, после которого идет название перечисления. Затем идет список элементов перечисления через запятую.

Перечисление - это упоминание объектов, объединённых по какому-либо признаку

Перечисления - это специальные классы, содержащие внутри себя собственные статические экземпляры.

Листинг 1: Пример перечисления

```
1 enum Season { WINTER, SPRING, SUMMER, AUTUMN }.
```

Когда мы доберёмся до рассмотрения внутренних и вложенных классов, в том числе статических, дополнительно это проговорим.

Перечисление фактически представляет новый тип данных, поэтому мы можем определить переменную данного типа и использовать её. Переменная типа перечисления может хранить любой объект этого исключения.

Season current = Season.SPRING; System.out.println(current);

Интересно также то, что вывод в терминал и запись в коде у исключений полностью совпадают, поэтому, в терминале мы видим ....

Каждое перечисление имеет статический метод values(). Он возвращает массив всех констант перечисления, далее мы можем этим массивом манипулировать как нам нужно, например, вывести на экран все его элементы.

Season[] seasons = Season.values(); for (Season s : seasons) System.out.printf("s s);

Именно в этом примере, я использую цикл foreach для прохода по массиву, для лаконичности записи. Чуть подробнее о его особенностях мы поговорим на одной из следующих лекций. Если коротко, данный цикл возьмёт последовательно каждый элемент перечисления, присвоит ему имя s точно также, как мы это делали в примере на две строки выше, и сделает эту переменную С доступной в теле цикла в рамках одной итерации, на следующей итерации будет взят следующий элемент, и так далее

Также в перечисления встроен метод ordinal() возвращающий порядковый номер определенной константы (нумерация начинается с 0).

System.out.println(current.ordinal())

Обратите внимание на синтаксис, метод можно вызвать только у конкретного экземпляра перечисления, а при попытке вызова у самого класса перечисления

System.out.println(Seasons.ordinal())

мы ожидаемо получаем ошибку невозможности вызова нестатического метода из статического контекста.

как мы с вами помним из пояснения связи классов и объектов, такое поведение возможно только если номер элемента как-то хранится в самом объекте. Мы видим в перечислениях очень примечательный пример инкапсуляции - мы не знаем, хранятся ли на самом деле объекты перечисления в виде массива, но можем вызвать метод вельюс. Мы не знаем, хранится ли в каждом объекте перечисления его номер, но можем вызвать его метод ординал. А раз перечисление - это класс, мы можем определять в нём поля, методы, конструкторы и прочее.

Перечисление Color определяет приватное поле code для хранения кода цвета, а с помощью метода getCode оно возвращается.

enum Color RED("#FF0000"), GREEN("#00FF00"), BLUE("#0000FF"); String code; Color (String code) this.code = code; String getCode() return code;

Через конструктор передается для него значение. Следует отметить, что конструктор по умолчанию приватный, то есть имеет модификатор private. Любой другой модификатор будет считаться ошибкой. Поэтому создать константы перечисления с помощью конструктора мы можем только внутри перечисления. И что косвенно намекает нам на то что объекты перечисления это статические объекты внутри самого класса перечисления. Также важно, что механизм описания конструкторов класса работает по той же логике, что и обычные конструкторы, то есть создав собственный конструктор мы уничтожили кон-

структор по-умолчанию, впрочем, мы его можем создать, если это будет иметь смысл для решаемой задачи.

Исходя из сказанного ранее можно сделать вывод, что с объектами перечисления можно работать точно также, как с обычными объектами, что мы и сделаем, например, выведя информацию о них в консоль

for (Color c : Color.values()) System.out.printf("s(s) c, c.getCode());

3.3.1. Задания для самопроверки

1. Перечисления нужны, чтобы: 3
   (a) вести учёт созданных в программе объектов;
   (b) вести учёт классов в программе;
   (c) вести учёт схожих по смыслу явлений в программе;
2. Перечисление - это: 2
   (a) массив
   (b) класс
   (c) объект
3. каждый объект в перечислении - это: 3
   (a) статическое поле
   (b) статический метод
   (c) статический объект

Практическое задание

1. Написать класс кота так, чтобы каждому объекту кота присваивался личный порядковый целочисленный номер.