#### **5.4 Records в Java 17**

### Что вы узнаете

В этом материале вы познакомитесь с возможностями нового типа классов — record, который появился в языке начиная с JDK 17. Вы узнаете, как сделать свой код понятнее и короче с помощью records, а также о их некоторых других полезных возможностях.

### Как работает record

В Java 17, помимо остальных важных улучшений, появилась новая форма записи класса — рекорд (англ. record). Рекорды удобно применять, если нужно быстро создать иммутабельный класс с возможностью быстрого доступа к полям.

Аббревиатурой POJO в Java называют классы, созданные для описания простых объектов. [POJO](https://ru.wikipedia.org/wiki/POJO" \t "/home/parviz/Documents\\x/_blank) (англ. Plain Old Java Object) — старый добрый Java-объект. Это Java-объект, который не наследуется от какого-то специфического класса и не реализует никакие служебные интерфейсы сверх тех, которые нужны для бизнес-модели. Также POJO, как правило, не несут какой-либо бизнес-логики.

Объекты нужны, чтобы хранить информацию в удобном для нас виде. Например, для описания студента мы можем ввести класс Student, у которого будут поля name, age и так далее. Типичный Java-объект выглядит так:

public class Student {  
  
  private final String name;  
  private final int age;  
  
  public Student(String name, int age) {  
    this.name = name;  
    this.age = age;  
  }  
  
  public String getName() {  
    return name;  
  }  
  
  public int getAge() {  
    return age;  
  }  
  
  @Override  
  public boolean equals(Object o) {  
    if (this == o)  
      return true;  
    if (o == null || getClass() != o.getClass())  
      return false;  
    Student student = (Student) o;  
    return age == student.age && name.equals(student.name);  
  }  
  
  @Override  
  public int hashCode() {  
    return Objects.hash(name, age);  
  }  
  
  @Override  
  public String toString() {  
    return "Student{" +  
        "name='" + name + '\'' +  
        ", age=" + age +  
        '}';  
  }  
}

В этом классе присутствует весь стандартный набор POJO: поля класса, конструктор, устанавливающий значения этих полей, метод toString() для корректного строкового вывода данных объекта, методы [equals() и hashCode()](https://www.internet-technologies.ru/articles/sravnenie-obektov-java-s-pomoschyu-equals-i-hashcode.html" \t "/home/parviz/Documents\\x/_blank) для сравнения объектов. Такой набор называется бойлерплейтом — это код, который программисты вынуждены писать в каждом подобном классе. Довольно утомительно, не правда ли?

Если вы знакомы с библиотекой Lombok, то знаете, что весь бойлерплейт можно сгенерировать автоматически с помощью аннотации ****@Value****. Разработчики Java 17 позволили нам отказаться от сторонних библиотек и перенести эту возможность в сам язык. С помощью рекордов запись подобного класса выглядит так:

 record Student(String name, int age) {}

Это намного короче и быстрее при написании.

### Для чего нужны records

В большей степени рекорды были добавлены для улучшения читаемости кода и уменьшения его количества.

Как вы узнали ранее, очевидные преимущества record — это автоматическое создание:

* конструктора класса со всеми аргументами (auto-generated public constructor);
* приватных финальных (private final) полей;
* методов hashCode(), equals() и toString();
* методов-геттеров для каждого поля.

Если вам нужно добавить в рекорд новое поле, все эти методы будут созданы автоматически — это очень удобно.

### Особенности использования record

Давайте посмотрим на records и узнаем, какие ещё возможности они нам дают.

****Возможность 1****: расширяем конструктор.

Если вам нужно добавить какую-то дополнительную логику в конструктор, это легко можно сделать:

record Student(String name, int age) {  
 public Student{  
  if (age < 0) {  
    System.out.println("Вы создали студента с отрицательным возрастом!");  
  }  
 }  
}

****Возможность 2****: добавляем собственный конструктор.

Иногда нам нужен ещё какой-то конструктор в дополнение к автоматически сгенерированному. Его можно сделать так:

record Student(String name, int age) {  
 public Student {  
    if (age < 0) {  
      System.out.println("Вы создали студента с отрицательным возрастом!");  
    }  
  }  
public Student (String name) {  
    this(name, 18);  
  }  
}

Обратите внимание: ваши конструкторы обязательно должны ссылаться на существующие с помощью ключевого слова ****this****.

****Возможность 3****: добавляем собственные методы класса.

Record, как и обычный класс, позволяет реализовывать методы внутри себя. Например, если нам нужно проверить возраст студента:

record Student(String name, int age) {  
 public boolean isAdult() {  
   return age >= 18;  
 }  
}

Однако если мы захотим добавить новые поля в рекорд подобным образом, то получим ошибку компиляции:

record Student(String name, int age) {  
  public boolean adult; *//вот так не сработает*  
*}*  
*}*

Для этого нужно добавить новое поле в объявление рекорда:

record Student(String name, int age, boolean adult) {}

****Возможность 4****: добавляем статические поля и методы.

Record в Java поддерживает создание статических полей и методов:

record Student(String name, int age) {  
  
  private static final int adultAge = 18;  
  
  public boolean isAdult() {  
    return age >= adultAge;  
  }  
  
  public static Student createAdultStudent(String name){  
   return new Student(name, adultAge);  
  }  
}

### Вывод

Рекорды в Java позволяют сильно сократить время, затраченное на написание стандартного кода, сэкономив его для других интересных задач. Советуем вам начать их использовать прямо сейчас!