UDC 004.932:528.854
DOI https://doi.org/10.32782/IT/2024-2-12

Leonid MESHCHERIAKOV
Doctor of Engineering, Professor, Department of Software Engineering, Dnipro University of Technology, 19, Dmytra Yavornytskoho ave., Dnipro, Ukraine, 49005, meshcheriakov.l.i@ nmu.one
ORCID: *0000-0002-9579-19701970*
Scopus-Author ID: *57205282540*

Volodymyr KUVAIEV
Doctor of Technical Sciences, Professor at the Department of Software Engineering, Dnipro University of Technology, 19, Dmytro Yavornytskyi ave., Dnipro, Ukraine, 49005, kuvaiev.v.m@nmu.one
ORCID: *0000-0001-6329-071X*
Scopus Author ID: *6602411915*

Alona KHAR
Assistant Lecturer at the Department of Software Engineering, Dnipro University of Technology, 19, Dmytro Yavornytskyi ave., Dnipro, Ukraine, 49005, khar.a.t@nmu.one
ORCID: *0000-0003-3176-7792*

Sergey DEMENKOV
Master of the Department of Computer System's Software, Dnipro University of Technology, 19, Dmytra Yavornytskoho ave., Dnipro, Ukraine, 49005, demenkov.s.o@nmu.one

To cite this article: Meshcheriakov, L., Kuvaiev, V., Khar, A., Demenkov, S. (2024). Avtomatychna heneratsiia kodu dlia ORM-systemy ActiveJDBC na bazi tekhnolohii annotation processin [Automatic code generation for ORM-system ActiveJDBC based on annotation processing technology]. *Information Technology: Computer Science, Software Engineering and Cyber Security*, 2, 97–102, doi: https://doi.org/10.32782/IT/2024-2-12

AUTOMATIC CODE GENERATION FOR ORM-SYSTEM ACTIVEJDBC BASED ON ANNOTATION PROCESSING TECHNOLOGY

The promising development of the software development field has led to the emergence of various templates and strategies for manipulating databases and, accordingly, the tools that implement them. One of the design patterns that is gaining more and more popularity is Active Record. This template allows encapsulating all the logic of working with the database table in the model class itself, which eliminates the need to create additional service classes that will be responsible for CRUD operations. One of the implementations of this approach is the ActiveJDBC ORM system. ActiveJDBC is developed as open source software and is a popular choice for developing Java applications that interact with relational databases.

The aim *of the work is to present the process of improving the user experience when working with the ActiveJDBC ORM system. The improvement is achieved through automatic code generation, which allows manipulating ActiveJDBC objects as POJOs.*

The methodology *for solving the task consists in the development of an annotation processor, which allows automatically code generating at the compilation stage that will interact with the ORM objects of the ActiveJDBC system.*

The scientific novelty *of the obtained results lies in the fact that for the first time software was developed that facilitates interaction with ActiveJDBC components and allows adding functionality that was not included in ActiveJDBC by its developer.*

Conclusions. *Approaches to the generation of program code in the Java programming language have been studied, and an annotation handler has been developed, which allows automatically at the compilation stage to generate code that will interact with ORM objects of the ActiveJDBC system. The developed tool allows: to free the programmer's working time from writing template code, and to focus on solving the tasks, and not on maintaining the database interaction tool; reduce the amount of code that must undergo validation during the Code Review process; increase the readability of the received code, thanks to the use of getters and setters, the names of which correspond to the JavaBeans specification, and the use of the Builder pattern.*

Key words: *ActiveJDBC, ORM, code generation, Java, Dynamic Proxy, Annotation Processor, database manipulation strategies.*

Леонід МЕЩЕРЯКОВ
доктор технічних наук, професор кафедри програмного забезпечення комп'ютерних систем, Національний технічний університет «Дніпровська політехніка», просп. Дмитра Яворницького, 19, м. Дніпро, Україна, 49005
ORCID: *0000-0002-9579-1970*
Scopus-Author ID: *57205282540*

Володимир КУВАЄВ
доктор технічних наук, професор кафедри програмного забезпечення комп'ютерних систем, Національний технічний університет «Дніпровська політехніка», просп. Дмитра Яворницького, 19, м. Дніпро, Україна, 49005
ORCID: *0000-0001-6329-071X*
Scopus Author ID: *6602411915*

Альона ХАРЬ
асистент кафедри програмного забезпечення комп'ютерних систем, Національний технічний університет «Дніпровська політехніка», просп. Дмитра Яворницького, 19, м. Дніпро, Україна, 49005
ORCID: *0000-0003-3176-7792*

Сергій ДЕМЕНКОВ
магістр кафедри програмного забезпечення комп'ютерних систем, Національний технічний університет «Дніпровська політехніка», просп. Дмитра Яворницького, 19, м. Дніпро, Україна, 49005

Бібліографічний опис статті: Мещеряков, Л., Куваєв, В., Харь, А., Деменков, С. (2024). Автоматична генерація коду для ORM-системи ActiveJDBC на базі технології Annotation Processor. *Information Technology: Computer Science, Software Engineering and Cyber Security,* 2, 97–102, doi: https://doi.org/10.32782/IT/2024-2-12

АВТОМАТИЧНА ГЕНЕРАЦІЯ КОДУ ДЛЯ ORM-СИСТЕМИ ACTIVEJDBC НА БАЗІ ТЕХНОЛОГІЇ ANOTATION PROCESSING

Перспективний розвиток галузі розробки програмного забезпечення призвів до появи різноманітних шаблонів та стратегій маніпуляцій з базами даних та, відповідно інструментів, що їх реалізують. Одним з шаблонів проектування, що набуває все більшої популярності є Active Record. Цей шаблон дозволяє інкапсулювати всю логіку роботи з таблицею бази даних у сам клас моделі, завдяки чому зникає необхідність створювати додаткові сервісні класи, які будуть відповідати за CRUD операції. Однією з реалізацій цього підходу є ORM-система ActiveJDBC. ActiveJDBC розвивається як відкрите програмне забезпечення і є популярним варіантом для розробки додатків на Java, які взаємодіють з реляційними базами даних.

***Метою роботи** є представлення процесу покращення користувацького досвіду при роботі з ORM-системою ActiveJDBC. Покращення достягається за рахунок автоматичної генерації коду, що дозволяє маніпулювати з ActiveJDBC об'єктами як з POJO.*

***Методологія** рішення представленого завдання складається в розробленні обробника анотацій, що дозволяє автоматично на етапі компіляції згенерувати код, що буде взаємодіяти з об'єктами ORM системи ActiveJDBC.*

***Наукова новизна** отриманих результатів полягає у тому, що вперше було розроблено програмне забезпечення, яке полегшує взаємодію з ActiveJDBC компонентами, та дозволяє додавати функціонал, який не був закладений у ActiveJDBC його розробником.*

***Висновки.** Досліджені підходи генерації програмного коду у мові програмування Java та розроблено обробник анотацій, що дозволяє автоматично на етапі компіляції згенерувати код, який буде взаємодіяти з об'єктами ORM системи ActiveJDBC. Розроблений інструмент дозволяє: вивільнити робочий час програміста від написання шаблонного коду, та зосередитись на вирішенні поставлених задач, а не на обслуговуванні інструмента взаємодії з базою даних; зменшити обсяг коду, що повинен пройти валідацію при проходженні процесу Code Review; підвищити читабельність отриманого коду, завдяки використанню геттерів та сеттерів, назви яких відповідають специфікації JavaBeans, та використанням патерна Будівельник.*

Ключові слова: *ActiveJDBC, ORM, генерація коду, Java, Dynamic Proxy, Annotation Processor, стратегій маніпуляцій з базами даних.*

The urgency of the problem. Undoubtedly, the success and growth of any IT project depends on many factors that can be considered at different levels: from strategic to operational. Accordingly, at the strategic level, the success of the project depends on how well it is aligned with the goals and needs of the organization that orders or implements it. On the operative level, it depends on how its main processes are performed: planning, organization, control and closure, as well as how correctly the main tools were chosen and how convenient it is to use them to achieve the operational and strategic success of the project.

It often happens that due to a not entirely successful decision when choosing a tool, software development requires writing a large amount of boilerplate code. This leads to the fact that the programmer spends a large part of his working time on maintaining the tool itself, which is used to solve the given task, and not on solving the task itself, which in turn leads to such consequences as: the company's funds are spent not on solving the set tasks, and to write templated, often repeated code; the likelihood of errors, bugs, and vulnerabilities increases, because template code may contain redundant, outdated, or incorrect parts that may cause conflicts, incompatibilities, or unsafe behavior; the size and complexity of the code base increases, as boilerplate code can take up a lot of space and resources, which can degrade software performance, speed, and reliability, and make such code significantly more difficult to read. And precisely because of this, in order to prevent the appearance of template code, tools are often used, the purpose of which, even at the project compilation stage, or already during the execution of the software code, is to generate code that the programmer would otherwise have to write himself. One example of a tool, the use of which can lead to the appearance of template code, is the ActiveJDBC ORM system, which is quite widespread due to its availability.

Analysis of recent research and publications. The existing development of the field of software development has led to the appearance of various templates and strategies for manipulating databases and, accordingly, tools that implement them. One of the design patterns that is gaining more and more popularity is Active Record. This template allows you to encapsulate all the logic of working with the database table in the model class itself, which eliminates the need to create additional service classes that will be responsible for CRUD operations. One of the implementations of this approach is the ActiveJDBC ORM system. ActiveJDBC is developed as open source software and is a popular choice for developing Java applications that interact with relational databases.

According to documentation (javalite.io) [1], the main principles of ActiveJDBC are as follows: extracting metadata from the database; configuration is based on agreements; no need to learn another query language. SQL will suffice; the code often reads like English text; no sessions, no attachment to sessions, no reattachment to sessions; no save managers; models are lightweight, simple POJOs; no Proxy. What is written is what is received; there are no getters and setters, but you can write them if necessary; no DAO and DTO; there is no anemic domain model.

Although ActiveJDBC declares that model classes must be POJOs, they are too lightweight, that is not a problem when using this tool in small projects, but as the codebase grows, it leads to a lot of boilerplate code. For example, if there is a need to access the attributes of a relation tuple, ActiveJDBC suggests using the «get» and «set» methods, and passing the name of the attribute itself to them, that can lead to programmer errors due to inattention and greatly complicates writing code, especially when in the project, which uses ActiveJDBC as an ORM uses data types not supported by ActiveJDBC. Examples of such types are java.time.LocalDate and java.time.LocalDateTime, which were added back in Java 8. One workaround is to use custom methods that will be added to ActiveJDBC that will do all the necessary conversions.

The next feature is that the base abstract class for ActiveJDBC Model objects does not implement the equals() and hashCode() methods, which makes it impossible to compare two ActiveJDBC objects with each other, and also makes it impossible to use such objects objects as keys for hash collections. Therefore, if they are needed, then again, users are forced to write them themselves.

Another potential limitation is that the toString() method is also not overridden, that can sometimes be inconvenient, for example, when outputing some information to the log is needed.

Also, it should be noted that another disadvantage of this tool is the inability to use the Builder spawning pattern to create new objects. Therefore, the development of a tool that will automatically generate template code for the ActiveJDBC ORM system remains relevant.

The purpose of the article is to present the process of improving the user experience when working with the ActiveJDBC ORM system. The improvement is achieved through automatic code generation, that allows manipulating ActiveJDBC objects as POJOs.

Presenting main material. Since the developed tool must be able to generate the code required by the user, first, it is necessary to find out how it is possible to achieve this goal in the Java programming language. Currently, there are several stages during which Java allows intervening in the program code and do something additional with it that was not explicitly specified by the programmer. They can be conventionally divided into: code generation during the execution of the program code, and code generation at the stage of compilation of the program code [2].

Runtime code generation is quite common and is used in many different frameworks and libraries. As an example of such a framework, the Spring Framework can be cited. According to a study conducted by B. Vermer (snyk.io) [3], who conducted a survey of more than two thousand respondents, Spring is used by 60% of them.

Code generation during execution can be conventionally divided into:

– code generation during class loading;

– code generation when calling the corresponding generator methods.

Both approaches use the ClassLoader as the main tool.

Compile-time code generation can be performed thanks to Maven plugins, but the annotation processing approach is usually used as the most versatile and independent means of automating work with Apache Maven software projects.

The annotation processor (Annotation Processor) is a tool in the Java development environment that allows developers to create their own annotations and define the logic associated with them. It plays an important role in modern Java development, allowing to automate many routine tasks and provide additional functionality for written code.

There is a loop followed by the javac compiler that allows each annotation handler to generate code for the annotations that are open to it. Here is the hierarchical order of this process (medium.com) [4]:

1. An element is marked with an annotation that is open to the annotation handler.

2. javac starts compiling the project classes. It is already aware of all existing annotation processors because it includes them in the classpath.

3. javac creates files with the extension «.class» during the first pass of processing. Since javac has already processed files with the «.java» extension, it already knows about the annotations that have been used. It then passes control to the appropriate annotation processor.

4. At this step, the annotation processor begins its work. Here it can generate additional files with the extension «.java» or do any other work.

5. At this point, all the initial «.java» files have been compiled, but the newly generated «.java» files have not yet been processed. However, javac will detect this and start another round of processing from the first step.

A schematic representation of the work of the annotation processor is presented in Fig. 1.

Annotation handlers are used in many popular frameworks and libraries such as Lombok, MapStruct and many others. They help to make the code more efficient, make fewer mistakes and allow you to create part of the functionality even at the compilation stage of the project.

It is relatively easy to start working with annotations. In order to create annotation processor, it is enough to implement the Processor interface or extend the AbstractProcessor class from the javax.annotation.processing package. An example of the code is shown in Fig. 2.

The results of the analysis of existing approaches to code generation are shown in Table 1.

Based on the data from the above table, it is concluded that to achieve the goal, the most optimal solution will be to use the annotation processing approach.

After the generation of wrapper classes for ActiveJDBC models, the following results were obtained depending on the number of lines of code that must be written without and with the use of a template code generator. The results are shown in Table 2.

Conclusions. Thus, approaches to the generation of program code in the Java programming language were investigated. An annotation handler was developed, which allows automatically generating code at the compilation stage that will interact with ORM objects of the ActiveJDBC system.

The developed tool allows to free the programmer's work time from writing template routine code, and to focus on solving the tasks set before him, instead of maintaining the database interaction tool; to reduce the amount of code that must undergo validation during the Code Review process; to increase the readability of the resulting code, thanks to the use of getters and setters whose names correspond to the JavaBeans specification, and the possibility of using the Builder pattern; to increase the stability of the received code due to the fact that the tool will automatically perform all necessary checks and transformations of data that will be received from the database or, conversely, stored in it; to prevent premature use of ActiveJDBC objects by using the Builder pattern.

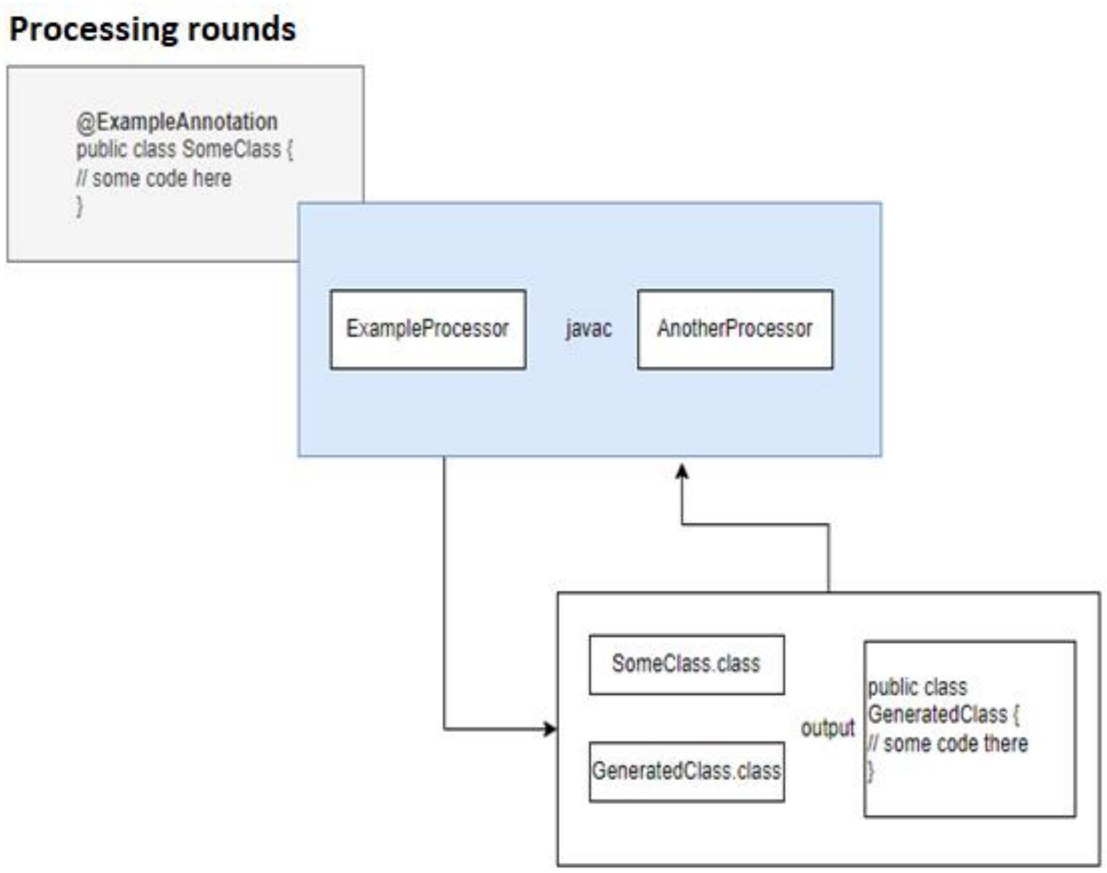
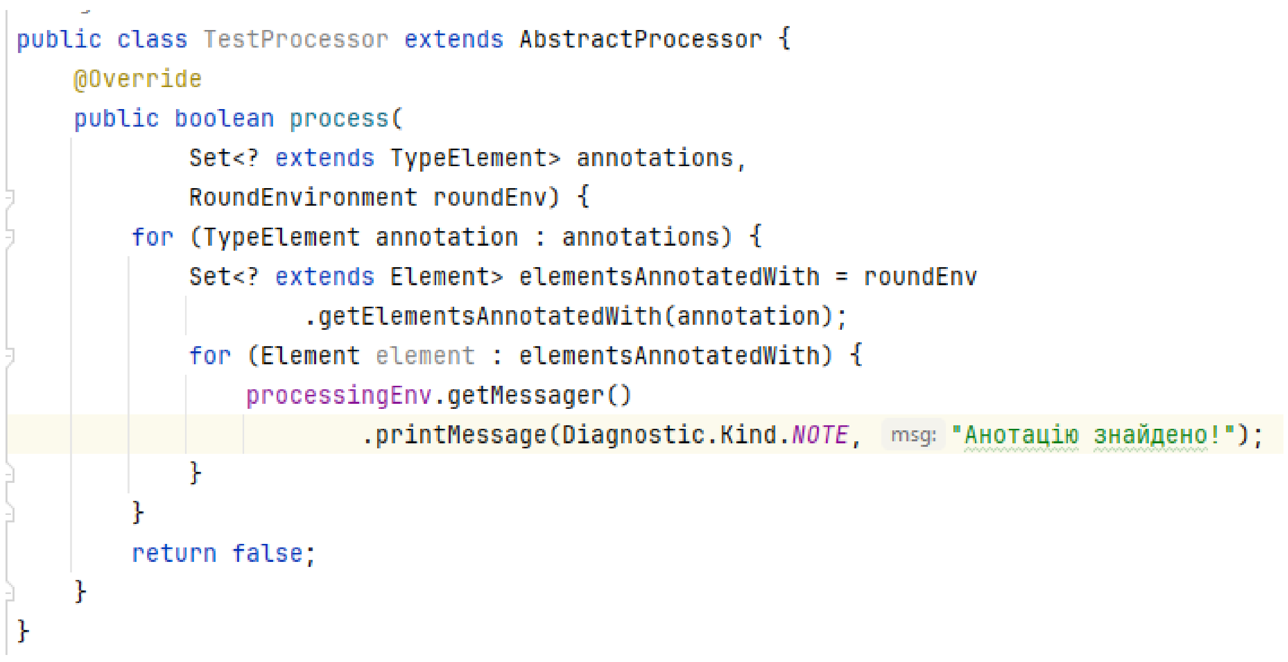


Fig. 1. Schematic representation of the work of the annotation processor



```
public class TestProcessor extends AbstractProcessor {
    @Override
    public boolean process(
            Set<? extends TypeElement> annotations,
            RoundEnvironment roundEnv) {
        for (TypeElement annotation : annotations) {
            Set<? extends Element> elementsAnnotatedWith = roundEnv
                    .getElementsAnnotatedWith(annotation);
            for (Element element : elementsAnnotatedWith) {
                processingEnv.getMessager()
                        .printMessage(Diagnostic.Kind.NOTE, msg: "Анотацію знайдено!");
            }
        }
        return false;
    }
}
```

Fig. 2. An example of an annotation handler

Table 1

Table of comparative characteristics of code generation methods

Characteristic	Code generation from runtime	Compile-time code generation
Creates an additional load on the compilation process	No	Yes
Creates an additional load on the system during code execution	Yes	No
Allows overriding the equals() method	Yes	Yes
Allows overriding the hashCode() method	Yes	Yes
Allows overriding the toString() method	Yes	Yes
Allows generating new methods	Yes, but this code cannot be used by a programmer when writing new code	Yes
Allows generating builder classes	Yes, but this code cannot be used by a programmer when writing new code	Yes

Table 2

Table of the dependence of the code terms number on the number of expected table attributes

Number of table attributes	Without using a boilerplate code generator and without the need for data transformation	Without using a boilerplate code generator and with data conversion	The number of code terms obtained using the boilerplate code generator
1	59	63	**12**
2	74	82	**13**
3	89	101	**14**
4	104	120	**15**
5	119	139	**16**
6	134	158	**17**
7	149	177	**18**
8	164	196	**19**
9	179	215	**20**
10	194	234	**21**

BIBLIOGRAPHY:

1. ActiveJDBC. URL: https://javalite.io/activejdbc (дата звернення: 28.10.2023).
2. JavaBeans(TM) Specification 1.01 Final Release. URL: https://download.oracle.com/otndocs/jcp/7224-javabeans-1.01-fr-spec-oth-JSpec/ (дата звернення: 28.10.2023).
3. Spring dominates the Java ecosystem with 60% using it for their main applications. URL: https://snyk.io/blog/spring-dominates-the-java-ecosystem- with-60-using-it-for-their-main-applications/ (дата звернення: 28.10.2023).
4. All About Annotations and Annotation Processors. URL: https://medium.com/swlh/all-about-annotations-and-annotation-processors-4af47159f29d (дата звернення: 28.10.2023).

REFERENCES:

1. ActiveJDBC. (n.d.). *javalite.io*. Retrieved from: https://javalite.io/activejdbc.
2. JavaBeans (TM) Specification 1.01 Final Release. (n.d.). *download.oracle.com.* Retrieved from: https://download.oracle.com/otndocs/jcp/ 7224-javabeans-1.01-fr-spec-oth-JSpec/
3. Spring dominates the Java ecosystem with 60% using it for their main applications. (n.d.). *snyk.io.* Retrieved from: https://snyk.io/blog/spring-dominates-the-java-ecosystem-with-60-using-it-for-their-main-applications/
4. All About Annotations and Annotation Processors. (n.d.). *medium.com.* Retrieved from: https://medium.com/swlh/all-about-annotations-and-annotation-processors-4af47159f29d