Списание „Математика, компютърни науки и образование“ Том 6, кн. 1, 2023
Journal of Mathematics, Computer Science and Education Vol. 6, No. 1, 2023

DOI: 10.54664/DQTC9983

ДИДАКТИЧЕСКИ БЕЛЕЖКИ ОТНОСНО ИЗПОЛЗВАНЕТО НА БЕЗИМЕННИ КЛАСОВЕ, НЕСТАТИЧНИ СТАРТИРАЩИ МЕТОДИ И ЗАПИСИ ПРИ ОБУЧЕНИЕТО В УВОДНИ КУРСОВЕ ПО ОБЕКТНО-ОРИЕНТИРАНО ПРОГРАМИРАНЕ С JAVA 21

Филип Петров

DIDACTIC NOTES ON USING UNNAMED CLASSES, NON-STATIC MAIN METHODS AND RECORDS IN INTRODUCTORY OBJECT-ORIENTED PROGRAMMING COURSES WITH JAVA 21

Philip Petrov

Abstract: *The article examines the potential usage of unnamed classes, non-static main methods and records in introductory Java object-oriented programming classes. It discusses the expediency of the technologies mentioned and provides some didactic notes for their fluent usage.*
Keywords: *OOP; Java 21; programming; didactics; teaching.*

ПРОБЛЕМЪТ С МНОЖЕСТВОТО НЕИЗУЧЕНИ ТЕРМИНИ В ПЪРВАТА ПРОГРАМА НА JAVA

Още от създаването на програмния език Java програмистите следват един и същи шаблон за създаване на нова програма със стартиращ (main) метод. Най-елементарният код, с който просто се изписва „Hello World“ в стандартния изход, изглежда по следния начин:

```
public class HelloWorld{
 public static void main(String[] args){
      System.out.println("Hello World");
 }
}
```

Основна критика от педагогическа гледна точка е наличието на прекалено много нови термини, които се струпват накуп пред обучавания. Намесват се модификатори за достъп (*public*); класове (*class*); статични методи (*static*); входни параметри на метод, в които клас за текстови низ (*String*) и масив ([]); изходни параметри (*void*); и извикване на метод от основната библиотека

(*System.out.println*). Този проблем е дискутиран в статии като [7] още в първите години, през които езикът все още е набирал популярност в образователните среди.

С въвеждането на интерактивния интерпретатор JShell с Java 9 през 2017 г. се появи възможност за изпълнение на команди без да се създава сорс файл с дефиниция на клас и main метод. Преподаватели по уводни курсове по програмиране започнаха да се възползват от този инструмент, за да може да накарат обучаваните да свикнат с основния синтаксис на езика и да усвоят основни знания на принципа „проба-грешка“. Чак след като се усвоят основни команди се преминава към и към създаване на по-сложни програми и изучаване на принципите на обектно-ориентираното програмиране (ООП). Пример за това е опита, който е споделен [9]. По-рано подобни експерименти са правени с други интерпретатори (например BeanShell, DynamicShell, и др.), но липсата на интеграцията им като част от стандартния Java Development Kit (JDK) прави затруднено използването им при извънкласна работа (нужно е следване на подробни стъпки за конфигурация, с които не всички начинаещи се справят безпроблемно). За преодоляване на този проблем традиционно са използвани онлайн компилатори. Пример за обучение с интерактивен интерпретатор в онлайн среда е показан в [5].

Въпреки простотата при използването на интерактивните интерпретатори или онлайн компилатори, при тях има един сериозен недостатък – изпълнението на програмния код команда по команда започва да прикрива цялостната му структура и взаимовръзката между различните елементи. Затова те са подходящи само за много малки чисто процедурни програми, но не и за код, който изисква проследяване на взаимовръзки между множество от обекти. В същото време смисълът от употребата на Java като език в уводен курс по програмиране е само ако този курс цели да запознае обучаваните с основите на ООП, а не за процедурно програмиране, за което има много по-подходящи езици. Поради тази причина употребата на интерактивни интерпретатори следва да се счита за ограничена и ако се използват, трябва да е сравнително кратко и фокусирано само като улеснение за по-лесно преминаване към писане на традиционен Java код.

ПРЕГЛЕД НА НЯКОИ СЪЩЕСТВУВАЩИ УЧЕБНИЦИ ЗА УВОДНИ КУРСОВЕ ПО ПРОГРАМИРАНЕ С JAVA

Традиционен ход на уводните курсове по програмиране с Java е да се направи преход с постепенно преминаване от процедурен стил на писане на код към обектно-ориентирано програмиране. Стандартно се започва с изучаване на примитивните типове данни, после се изучават някои основни вградени обекти, изучават се основни езикови конструкции като условен оператор и цикъл, масиви, методи, а чак след това се разглежда създаването на собствени класове и обекти. Това е естествено и нормално, защото по този начин понятията се въвеждат постепенно, а знанията се подреждат стъпаловидно и във възходяща сложност. Подобна структура се следва от повечето учебници по ООП с Java. Такива примери са [1], [3], [8], [10], [11] и много други, при които дефинирането на собствени класове и изучаването на основните концепции на ООП започва след първата третина или близо до средата на учебника. Java е много подходящ език за такъв стил на въвеждане на знанията, защото въпреки, че е обектно-ориентиран език, той включва много елементи от процедурно програмиране. При други по-изчистени в концептуално ниво езици като например SmallTalk това по-скоро не би било възможно, защото там няма примитивни типове и няма езикови конструкти като условен оператор и цикъл, а всичко е реализирано като обекти и предаване на съобщения между тях, т.е. там понятията *обект* и *съобщение* е важно да бъдат изучени от самото начало. В този смисъл Java може да се разгледа и като мултипарадигмен език, с който може да се преподава както процедурно, така и обектно-ориентирано.

Само при някои отделни учебници по Java могат да се наблюдават изключения от споменатата структура. В [6] авторът полага сериозно усилие да обясни в теоретичен план концепциите на ООП преди да се започне демонстрацията на какъвто и да е програмен код. Въпреки това впоследствие книгата продължава по традиционен начин и сериозна работа с класове и обекти се появява чак във втори том. Малко са учебниците като [2], в които от самото начало се набляга директно

на писането на код, с който се дефинират собствени класове със съответни конструктори, get и set методи, и т.н. В него може ясно да се забележи основният недостатък от дидактическа гледна точка, а именно че в началото обучаваните са претрупвани с огромно количество нови термини, повечето от които не се упражняват и съответно не се усвояват непосредствено след дефиницията им. Това е методически недостатък, който нарушава принципа за стъпаловидно натрупване на знанията.

В [4] е проучена възможността за ранно въвеждане на понятията *клас*, *обект*, *наследяване* и *полиморфизъм* в училищния курс по информатика, като за подходът със започване на изучаване на обекти в самото начало (objects-first) авторите обобщават следното: *за да могат обучаемите да експериментират на този ранен етап, те трябва да използват готови или полуготови интерактивни програми*. Като основен недостатък на този подход авторите посочват, че *основните елементи, чрез които се реализират алгоритмите (променливи, изрази, условна логика, цикли и др.), са засенчени от допълнителните нива на абстракция, които внасят класовете и обектите*. Това означава, че поне първите теми от учебниците, по които се преподава в този стил, трябва да включват само насоки и пропедевтика към бъдещо задълбочено изучаване на съответните знания, а не да се разглеждат знанията задълбочено и изчерпателно. Учебниците за училищния курс по информатика определено спазват това правило. Според действащата учебна програма по информатика за VIII клас се започва директно с изграждане на приложения с графичен потребителски интерфейс, т.е. учениците работят активно с множество класове и обекти още в самото начало, но започват да проучват тяхната същност и създават свои собствени класове едва към края на курса.

Вижда се, че проблемът с претрупване с нови термини, част от които остават неизучени или неупражнени за много продължителен период от време, е основен при уводните курсове по програмиране с Java. Това е и основната критика, която често се използва като предпоставка за избор на друг език за курсовете по Увод в програмирането във висшите училища. Основният начин за смекчаването на този проблем се е утвърдил чрез започване на курса с догматично въвеждане на термини, които не се изучават, преподаване в процедурен стил на програмиране с плавно надграждане към концепциите на ООП, а при първите часове евентуално се употребяват интерактивни интерпретатори преди да се премине към пълнофункционалната среда за разработка на софтуер и писане на по-сериозни програми. Тези решения по-скоро се отчитат от преподавателите като недостатъчни.

БЕЗИМЕННИ КЛАСОВЕ, НЕСТАТИЧНИ СТАРТИРАЩИ МЕТОДИ И ЗАПИСИ

С Java 21 са въведени две нововъведения – безименни класове (*unnamed classes*, което не трябва да се бърка с анонимни класове) и нестатични стартиращи методи (*instance main methods*). Както подсказват самите им имена, първото е свързано с възможност да се пропуска задаването на име на клас (от примера в началото на статията: *public class HelloWorld*), когато се изпълнява единичен самодостатъчен файл, а второто дава възможност за нестатични стартиращи методи без входни параметри. На практика това означава, че при програмата с изписване на «Hello World» в стандартния изход вече може да се пропуснат повечето от все още непознатите термини:

```
void main(){
 System.out.println("Hello World");
}
```

Безименният клас се слага в самостоятелен файл и за него няма дефиниран пакет. Извън обучението те биха получили приложение изключително рядко, защото не се предполага да са част от сериозен софтуер. Реално това, което прави компилатора при засичане на анонимен клас с нестатичен стартиращ метод, е да вкара кода в следния контекст (името на класа се взима автоматично според името на файла, в който е записан кода – в примера HelloWorld.java):

```
public class HelloWorld {
    public static void main(String[] args) {
        new Object() {
            void main(){
                System.out.println("Hello World");
            }
        }.main();
    }
}
```

Това създаване на анонимен обект и прикриването на основната структура на програмата остават скрити за разработчика (в случая обучавания). От дидактическа гледна точка тези подробности не е нужно да се разясняват нито в началото, нито дори в края на уводния курс по програмиране, защото съкратения запис не се очаква да се използва в бъдеще. Идеята за стартиране на обучението по ООП с Java с безименни класове и нестатични стартиращи методи не е да се опрости езика за програмиране като цяло, а само да се улеснят встъпителните часове в обучението, докато се усвоят основни команди за работа с примитивни типове данни, изучат се някои методи от клас String и се дадат поне няколко примера за условен оператор и цикъл. По-нататък съкратеният запис следва „да се разрастне“ до стандартния синтаксис за дефиниране на клас със стартиращ метод, откъдето да започне добавянето на собствени класове и съответно изучаването на основните принципи на ООП (абстракция, капсулация, наследяване и полиморфизъм). От чисто дидактическа гледна точка е важно как точно да се осъществи този преход.

Ако чрез нестатични стартиращи методи се последва традиционния ход на курса, а именно започне да се въвежда писане на собствени методи, а чак след това въвеждане на класове, ще се породи нов методически проблем. Когато кодът се пренесе към пълния запис, ще се окаже, че извикванията към нестатични методи няма да работят. Например ако обучаваният е направил следния код в безименен клас:

```
void main(){
      System.out.println(sum(2,3));
}
int sum(int a, int b){
      return a+b;
}
```

пренасяйки го механично в статичен контекст, без наличие на нужното задълбочено разбиране на това как са реализирани безименните класове и нестатичните стартиращи методи, ще се получи грешка *non-static method sum(int, int) cannot be referenced from a static context*:

```
public class HelloWorld {
    public static void main(String[] args) {
        // този код няма да може да се компилира:
        System.out.println(sum(2,3));
    }
    int sum(int a, int b){
        return a+b;
    }
}
```

Решението тук действително е единствено с добавяне на единствената ключова дума *static* за метод *sum*, но тя не се появява по интуитивен път, поради което би изглеждала странна и обър-

кваща за обучаваните. В същото време обяснението какво е статичен и какво нестатичен метод няма особен смисъл преди да бъдат обяснени по-подробно понятията *клас* и *обект*. При това дори да се даде подобно обяснение на чисто концептуално ниво, отново ще остане неизяснен въпросът „защо преди работеше, а сега не работи?“.

Поради изложената причина от методическа гледна точка е целесъобразно кодът да се разшири първо в посока на въвеждане на понятието *клас* и чак след това към изясняване на мястото и ролята на методите в класовете. Много подходящи за този етап от обучението са записите (*record*). Те са непроменими (immutable) класове, които са въведени с Java 16. При тях може да се пропусне дефинирането на конструктор, защото генерират наготово такъв с подадени параметри. Идват с готов набор от реализирани *get* методи, както и стандартни методи *hashCode* и *equals*. Например запис *Dog* с член променливи *name*, *breed* и *age* минималистично би изглеждал по следния начин:

```
record Dog(String name, String breed, int age) {}
```

Създаването на обекти се прави по традиционния начин както при обекти, дефинирани чрез нормални класове. Освен това при записите е много лесно се добавят методи:

```
void main() {
      Dog charlie = new Dog("Charlie", "Cavalier", 2);
      charlie.bark();
      Dog sharo = new Dog("Sharo", "Unknown", 7);
      sharo.bark();
}

record Dog(String name, String breed, int age) {
      void bark() {
      System.out.println(name + ": Woof");
}
}
```

Горният код би могъл да бъде преработен директно към „пълния синтаксис“ без да се получи споменатия преди това проблем:

```
public class Test {
      public static void main(String[] args) {
            Dog charlie = new Dog("Charlie", "Cavalier", 2);
            charlie.bark();
            Dog sharo = new Dog("Sharo", "Unknown", 7);
            sharo.bark();
      }

record Dog(String name, String breed, int age) {
            void bark() {
                  System.out.println(name + ": Woof");
            }
      }
}
```

Естествено би възникнал въпросът, че така обучаваните се заблуждават и се създава погрешна представа за равнозначност между двата програмни кода, а те очевидно не са еквивалентни. Въпросът е дискусионен, но предвид това, че съкратения запис по-скоро не се очаква да

бъде използван в бъдеще, преподавателите могат да премълчат за разликите и да ги оставят за изясняване за по-късен етап, т.е. чак след като вече бъдат изучени в детайли концепциите на ООП.

Употребата на *record* вместо стандартен клас е препоръчително допълнително опростяване на програмния код, което следва плътно споменатата концепция за облекчаване на кода и намаляване на непознатите понятия. Ако трябва подобен пример да се реализира със стандартен клас, ще бъде нужно като минимум да се напише тривиален конструктор с подадени параметри, от който обикновено се налага и изучаването на препратките към член-променливи чрез ключовата дума *this*. Вместо това чрез *record* може да се постигне по-плавно надграждане на знанията, като първоначалния кратък и минималистичен код започне да се разширява до негов аналог към обикновен клас. Това разширение също се извършва постепенно. Първоначално може да се покажат допълнителни проверки за валидност на данните чрез конструктор в самия *record*, с които да се извършва пропедевтика за капсулацията на данни и да се загатне механизма за работа с изключения:

```
record Dog(String name, String breed, int age) {
    public Dog{
        if(age<0) throw new IllegalArgumentException("Age<0");
    }
    void bark() {
        System.out.println(name + ": Woof");
    }
}
```

След това знанието може да се разшири чрез създаване на нормален клас, където по аналогия с вече създаден record (но без еквивалентност между двете – нещо, което трябва да бъде казано и евентуално обяснено подробно на по-късен етап) се показва дефинирането на член-променливи и конструктор с подадени параметри:

```
class Dog {
    String name;
    String breed;
    int age;

    public Dog(String name, String breed, int age) {
        if (age < 0) {
            throw new IllegalArgumentException("Age<0");
        }
        this.name = name;
        this.breed = breed;
    }

    void bark() {
        System.out.println(name + ": Woof");
    }
}
```

Обучението може да продължи като се покажат допълнителни нови възможности, като например конструктор по подразбиране и др. След достатъчно упражнение и решаване на различни примери със създаване на различни класове, ще може да се пристъпи към изясняване на модификаторите за достъп, извършване на капсулация с *get* и *set* методи и евентуално накрая да се обобщи същността на непроменимите (immutable) обекти, които в началото са създавани чрез *record*.

С показаната техника записите се явяват лесен за ползване инструмент заради своя кратък код. Чрез тях може последователно да се въвеждат нови езикови конструкции и да се изучават

нови понятия. Чрез постепенното разкриване на същността на *record* изучаването на капсулацията протича плавно и дава възможност за поставяне на обучавания в проблемни ситуации, в които е поставена ясна цел и е наличен пример какво точно трябва да се постигне. Още по-съществено е, че новите понятия се въвеждат бавно, последователно и след достатъчно упражнение на предишни, вече изучени опорни знания. При това не е нужно да се отива до крайност. Някои от получените наготово методи в *record* като например *hashCode* и *equals* може да се изясняват на много по-късен етап или дори да се пропуснат напълно при уводния курс.

ОГРАНИЧЕНИЯ И ТЕКУЩИ ЗАТРУДНЕНИЯ

Записите са вече приета технология и се използват от версия 16 на стандартния JDK и се поддържат безпроблемно от всички популярни среди за разработка. Безименните класове и нестатичните стартиращи методи към версия 21 все още са режим за предварителен преглед. Това означава, че тяхното пълно въвеждане в езика ще бъде направено евентуално при следваща версия. Поради тази причина употребата на тези нововъведения при версия 21 ще изисква компилирането и изпълняването на кода с добавяне на допълнителни флагове, които затрудняват процеса за стартиране на програмата. Например ако кодът е записан във файл с име hello.java, неговата компилация при JDK21 трябва да се извърши със следната команда:

```
javac --enable-preview --release 21 hello.java
```

Изпълнението на генерирания test.class файл от своя страна ще трябва да се направи чрез следната команда:

```
java --enable-preview hello
```

Освен това на този етап не е възможно използването на техниката в среди за интегрирана разработка (IDE) на софтуер като Netbeans 19 и Eclipse 2023-09, защото при тях не е предвидено изпълнение на самостоятелни файлове (задължително е да се създаде проект), а дори при добавяне на файл в проект и настройка за компилационен флаг --enable-preview средите все още отказват да изпълнят код с безименни класове и нестатични стартиращи методи. На този етап не е ясно дали нововъведенията ще бъдат въведени във въпросните среди в бъдеще, дори да бъдат приети като стандартна част от езика.

Последният проблем не е задължително недостатък и препятствие пред предложените технологии. От методическа гледна точка дори е добре първоначално да се покаже процесът на компилиране на java файлове и изпълнение на class файлове чрез съответните програми през командния ред. В началото обучението е препоръчително да започне без употребата на графични среди за разработка на софтуер, а вместо това кода да се пише в най-обикновен текстови редактор. Така ще се покаже на обучаваните, че компилирането и изпълнението са отделни поредни и свързани действия – нещо, което в средите е прикрито зад еднократно натискане на един графичен бутон (възможностите за отделно компилиране и изпълнение са възможни през меню, но по подразбиране не са част от стандартната лента с бутони). Освен това е препоръчително да се покаже в явен вид и да се изясни от самото начало, че тези среди са само удобство и улеснение за писане на програмен код, а не са задължителна част при разработката на програми. В училище например не е рядкост да се наблюдава известно объркване сред учениците и някои от тях да казват, че учат „програмиране на Netbeans“, вместо „програмиране на Java“. Не на последно място може да се отбележи, че популярните среди са с много претрупан графичен интерфейс, който първоначално може да изглежда прекалено сложен за обучаваните. Затова написването на първите няколко програми с обикновен текстови редактор и компилирането им през команден ред е по-скоро препоръчително във всички случаи, т.е. дори при традиционното обучение без безименни класове и нестатични стартиращи методи.

ЗАКЛЮЧЕНИЕ

Безименните класове, нестатичните стартиращи методи и записите са удобен дидактически инструмент за постепенно и плавно въвеждане на нови понятия в обучението по обектно-ориентирано програмиране с Java. С тях е сравнително лесно да бъдат прикрити много от езиковите конструкции и термини, които първоначално затормозяват начинаещите и правят кода на програмите дълъг и труден за усвояване. Използването на тези инструменти трябва да се прави с нужното внимание – преподавателите следва да са предпазливи в процеса на надграждане на знанията, за да избегнат нежелано объркване. Предстои да се разбере дали нововъведенията от версия 21 ще бъдат утвърдени и въведени официално в езика в бъдещи версии на езика. Ако това се случи, авторът препоръчва на преподавателите да се възползват от новите възможности и да преструктурират встъпителните си уроци.

БЛАГОДАРНОСТИ

Изследването е подкрепено от проект №80-10-61/25.4.2023 г. (Организационни форми за професионална квалификация на педагогическите специалисти в обучението по математика, информатика и информационни технологии) към Фонд научни изследвания на Софийски университет „Св. Климент Охридски“, 2023–2024 г.

ЛИТЕРАТУРА

[1] Василев, А. 2020. Java за всички. *Асеневци*, София. // Vasilev, A. 2020. Java for all. *Asenevtzi*, Sofia.

[2] Кръстев, Е. 2017. Увод в програмирането. *Университетско издателство „Св. Климент Охридски“*, София. // Krastev, E. 2017. Introduction to Programming. *University Press “St. Kliment Ohridski”*, Sofia.

[3] Наков, С., и др. 2008. Въведение в програмирането с Java. *Национална академия за разработка на софтуер*, София. // Nakov, S. 2008. Introduction to Programming with Java. *National Academy for Software Development*, Sofia.

[4] Чотова, Г., Дончев, И. 2019. Реализация на objects-early подход в училищния курс по информатика. *Списание „Математика, компютърни науки и образование“*, том 2, брой 2, Велико Търново. // Chotova, G., Donchev, I. 2019. Objects-Early Approach to Teaching Informatics at School Level. *Journal “Mathematics, Computer Science and Education”* vol. 2, No. 2.

[5] Bieg, C., & Diehl, S. 2004. Educational and technical design of a Web-based interactive tutorial on programming in Java. *Science of Computer Programming*, 53(1), 25-36.

[6] Eckel, B. 2000. Thinking in Java, 2nd edition. *Prentice Hall Inc*.

[7] Hong, J. 1998. The use of Java as an Introductory Programming Language. *XRDS: Crossroads, The ACM Magazine for Students*, 4(4), 8-13.

[8] Horstmann, C. 2000. Computing Concepts with Java Essentials. *John Wiley & Sons Inc*.

[9] Kendall-Morwick, J. 2020. Using JShell in CS1. *Journal of Computing Sciences in Colleges*, 35(6), 84-91.

[10] Roberts, S., Heller, P., Ernst, M. 1999. Complete Java2 Certification Study Guide. *SYBEX inc*.

[11] Schildt, H. 2001. Java 2: A Beginner Guide. *McGraw-Hill Osbourne Media*.

ИНФОРМАЦИЯ ЗА АВТОРА

доц. д-р Филип Петров Петров, Факултет по математика и информатика, Софийски университет „Св. Климент Охридски“, philip@abv.bg

ABOUT THE AUTHOR

Philip Petrov – Associate Professor, PhD, Faculty of Mathematics and Informatics, St. Kliment Ohridski University of Sofia, e-mail: philip@abv.bg