Л Е К Ц И Я №1

Основи на програмния език Java

Оператори и конструкции

I. ВЪВЕДЕНИЕ

Идеята за създаването на програмния език Java възниква по време на работа по проекта **Stealth** на Sun Microsystems в началото на 1991г. Целта на проекта е била разработването на "интелигентни" електронни устройства, които могат да бъдат програмирани и управлявани дистанционно. Идеята за надеждна работа на множество устройства, изградени въз основа на различни апаратни платформи и функциониращи в мрежова среда, спомага по-късно Java технологиите да са предпочитани при създаването на мрежови приложения. Основните участници в проекта Stealth са: Бил Джой (B. Joy), Патрик Нотън (P. Naughton), Майк Шеридан (M. Sheridan) и Джеймс Гозлин (J. Gosling). През 1991г. проектът Stealth се преименува на Green Project. За изборът на подходящ програмен език в рамките на **Green Project** отговаря Гозлин. Първоначално той решава да използва C++, но бързо се отказва - C++ не е подходящ за разработка на високо надеждни, платформено независими приложения с компактен код. Използвайки C++, през 1992г. Гозлин създава нов програмен език, който той нарича **Oak** (от прозореца на офиса на Green Project в Menlo Park се е виждал дъб). През лятото на 1992г. колективът на Green Project е готов да демонстрира първите постигнати резултати - разработен е прототип на миниатюрен контролер за отдалечено управление на домашна електроника. Апаратната част на контролера с име *7 (Star Seven) е разработена от Ед Франк (E. Frank), който се включва към Green Project през април 1991г. Програмното осигуряване е изцяло написано на Oak. Гозлин разрешава проблема с изискването за платформена независимост, като направи Oak *интерпретаторен* език. За целта програмите на Oak се компилират до междинен код, наречен *byte code*. Той не може да се изпълнява от конкретен микропроцесор. За да се получи машинен код за конкретен микропроцесор е необходима т.нар. *виртуална машина* (VM), която преобразува byte code до желания машинен език. През ноември 1992г. Green Project е преименуван на **FirstPerson**. Усилията на колектива, формиращ FirstPerson, е използването на Oak при интерактивните телевизионни технологии, например видео по поръчка. Целта обаче не се постига и Sun Microsystems са принудени да пренасочат голяма част от членовете на FirstPerson на друга работа. Спасението на Oak се оказват Интернет технологиите. През 1994г. Бил Джой стартира проекта **LiveOak**, чиято задача е разработването на малка по размер, но с големи функционални възможности операционна система. През юли 1994г. П. Нотън решава да насочи функционирането на LiveOak в Интернет пространството. За да се докажат функционалните възможности на Oak за работа в мрежова среда се решава да се разработи WEB браузър, който да е конкурент на излезлия една година по-рано на пазара браузър Mosaic. През септември 1994г. П. Нотън и Джонатан Пейн (J. Payne) успяват да напишат кода на браузър, наречен **WebRunner**, който по-късно се преименува на **HotJava**. През октомври са първите демонстрации на HotJava. Браузърът може да изпълнява Oak код под формата на **аплети**. Това позволява рязко подобряване на интерактивността на WEB страниците.

За рождена дата на Java се счита 1995г., когато програмният език Oak се преименува на Java. Това се налага, тъй като се оказва, че наименованието Oak е невъзможно да се използва, защото вече има регистриран програмен език с това име. През същата година се анонсира на пазара Java версия 1.0. Последната версия е Java SE 14, която се анонсира през март 2020. Версиите, които основно се използват от Java програмистите, са:

- Java SE 8 (анонсира се през март 2014 и ще се поддържа до декември 2030).
- Java SE 11 (анонсира се през септември 2018 и ще се поддържа до септември 2026).

Развойни среди (IDE) за Java

На съвременния етап не е възможно разработване на софтуер без използване на развойна среда. Развойните среди могат да бъдат използвани както off-line, така и on-line. На Фиг. 1.1 са показани трите най-високо оценени от програмистите на Java развойни среди.

Java IDE	User Rating	User Satisfaction	Learning Curve Scale	Syntax Highlighting	Performance
Eclipse	4.8/5	92 %	Easy	Yes	Good
IntelliJ Idea	4.3/5	89 %	Medium	Yes	Average
NetBeans	4.1/5	85 %	Medium	No	Average

Фиг. 1.1. Класация на развойните среди за Java, според удовлетвореността на програмистите

1. Eclipse
Развойната среда Eclipse се анонсира през 2001г. Може да се използва със следните операционни системи: Windows, Linux, macOS и Solaris. Облачното базирания вариант на Eclipse, наречен Eclipse Che, позволява на програмистите да разработват приложения чрез уеб браузър. Eclipse поддържа разработването на приложения на няколко програмни езика чрез използване на плъгини: C, C++, Clojure, Groovy, Haskell, JavaScript, Julia, Perl, PHP, Ruby, Rust и Scala.

2. IntelliJ IDEA
Развойната среда IntelliJ се анонсира през 2001г. Може да се използва със следните операционни системи: Windows, Linux и macOS. Издава се под лиценз на Apache 2 и е патентовано търговско наименование. Тази развойна среда разполага с функции за рефакториране на различни езици и анализ на потока данни. Тя осигурява поддръжка за други базирани на JVM програмни езици, например Kotlin.

3. NetBeans
Развойната среда NetBeans се анонсира през 1997г. Може да се използва със следните операционни системи: Windows, Linux, macOS и Solaris. Фирма Oracle анонсира NetBeans като официален IDE за Java SE 8. Развойната среда NetBeans е с отворен код и улеснява програмистите на Java да изграждат настолни, мобилни и Web приложения. За да проектира и разработи GUI за Java SE, NetBeans предлага GUI Builder. Специализираното

Java IDE е достъпно на 28 различни езика. NetBeans има разширения, които позволяват да се използват следните програмни езици: C, C++, HTML5, JavaScript, PHP и др.

Коя развойна среда да използвате?

Всяка една от анализираните развойни среди има както предимства, така и недостатъци. Най-мощна като функционалност е Eclipse. Ако използвате и други програмни езици, а не само Java тази среда е подходяща за вас. Облачно базирания вариант позволява писане на Java код и за специализирани апаратни палтформи (вградени и IoT системи).

Програмната среда NetBeans е най-старата от трите IDE. Трябва да се отбележи, че Oracle са избрали тази среда за официална IDE за Java 8. Може да създавате всички видове приложения, които Oracle Java позволява.

Най-новата развойна среда от изброените е IntelliJ. Те е тясно специализирана за разработка на Java приложения, или приложения, които се базират на език, който използва JVM. Средата е разработена така, че максимално да улесни програмиста.

За нуждите на курса на обучение няма значение коя развойна среда ще използвате, тъй като ще създавате приложения с ниска сложност. В лаборатория 3304 е инсталирана развойна среда NetBeans, тъй като това е официалния избор на Oracle и има добра поддръжка и на останалите технологии, които ще използваме: HTML5, CSS3 и JavaScript.

II. РЕЙТИНГ НА JAVA

Всяка класация на програмни езици е специфична, тъй като се реализира на базата на специфични критерии, например: брой разработени комерсиални приложения; използване на програмния език в сферата на обучението и науката; брой създадени проекти с този програмен език; средна заплата на програмистите, които използват този език.

Независимо от типа на класацията, за последните 10 години Java винаги е в топ 3 на по-реномираните класации. Докато преди 2-3 години Java нямаше конкуренти, то през 2019-2020 се забелязва ръст на програмни езици като Python и JavaScript. Причината за ръста при Python е неговото широко използване както при обучение, така и комерсиално, основно в областта на изкуствения интелект. Ръстът на JavaScript се дължи на чисто комерсиални причини. Основната причина за това е големият брой на Web приложенията и най-вече full-stack JavaScript приложенията (използване на JavaScript както за front-end, така и за back-end програмиране).

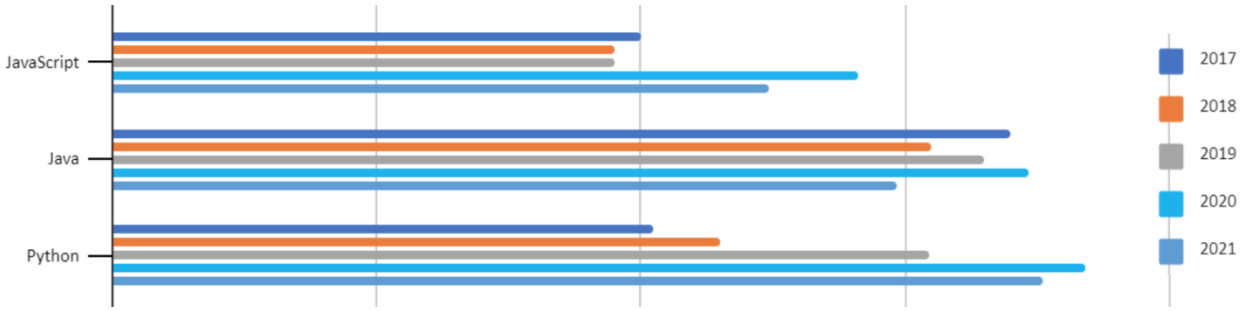
На Фиг. 1.2 е показана класация на програмните езици за 2021. Python е водещ програмен език при класацията на IEEE, Tiobe Index и Coding dojo. В тези класации Java е на второ място, а JavaScript на трето и пето място. От гледна точка на проектите в Github на първо място е JavaScript. От тази класация ясно се вижда тенденцията за висок ръст на JavaScript и Python проектите, докато Java има стабилно представяне за последните 4 години. Тенденцията е все по-широко използване на програмните езици, които са платформено независими (Java), на тези чрез които се създават full-stack Web и облачни услуги (JavaScript) и тези, които са свързани със създаване на приложения в областта на изкуствения интелект (Python).

# Ranking	Programming Language	Percentage (YoY Change)
1	JavaScript	19.014% (+0.225%)
2	Python	16.351% (+0.243%)
3	Java	12.817% (+2.086%)
4	Go	7.574% (-1.348%)
5	Ruby	7.194% (+0.702%)
6	TypeScript	7.183% (-0.151%)
7	C++	6.695% (-0.941%)
8	PHP	4.910% (-0.288%)
9	C#	3.413% (-0.383%)
10	C	3.029% (-0.292%)

a) GitHub

1	Python	100.0
2	Java	95.3
3	C	94.6
4	C++	87.0
5	JavaScript	79.5
6	R	78.6
7	Arduino	73.2
8	Go	73.1
9	Swift	70.5
10	Matlab	68.4

b) IEEE



c) codingdojo

Aug 2021	Change	Programming language	Share	Trends
1		Python	29.93 %	-2.2 %
2		Java	17.78 %	+1.2 %
3		JavaScript	8.79 %	+0.6 %
4		C#	6.73 %	+0.2 %
5	↑	C/C++	6.45 %	+0.7 %
6	↓	PHP	5.76 %	-0.0 %
7		R	3.92 %	-0.1 %
8		Objective-C	2.26 %	-0.3 %
9	↑	TypeScript	2.11 %	+0.2 %
10	↓	Swift	1.96 %	-0.3 %

d) Tiobe index

Фиг. 1.2. Класация на програмните езици за 2021

III. ВЪВЕДЕНИЕ В JAVA

Java е *технология* на Sun Microsystems, която е както обектно-ориентиран програмен език от високо ниво, така и програмна платформа. Java *платформата* се състои от два компонента:

- Java виртуална машина, Java Virtual Machine (JVM);
- Java приложни програмни интерфейси, Application Programming Interfaces (API).

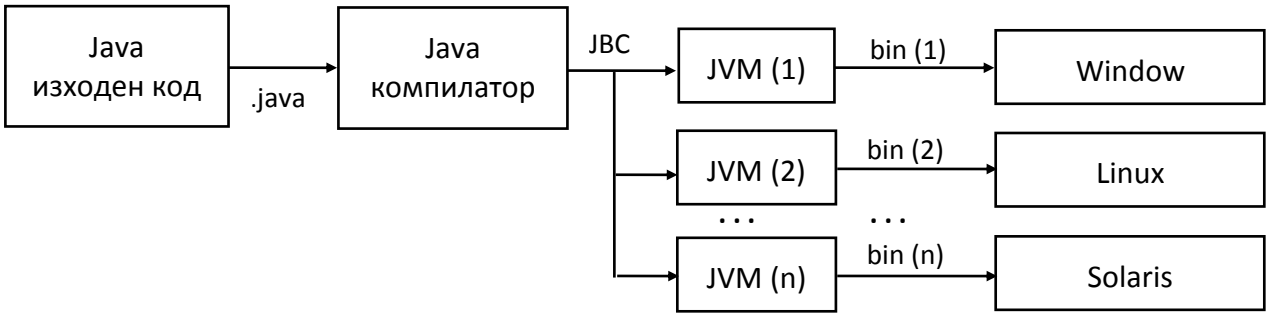
Java API са съвкупност от готови за използване програмни компоненти, които формират библиотеки. Те се включват към кода на даден проект чрез специална ключова дума - import. По-голямата част от платформите комбинират необходимата апаратна част и операционната система (ОС), под управлението на която се стартират разработените приложни програми. Java платформата, известна още като Java Runtime System (JRS), е чисто програмна и това позволява Java приложенията да са независими от апаратните платформи на които се стартират.

Основните *характеристики* на програмния език Java са:

- обектно-ориентиран;
- платформено независим;
- надеждна обработка на изключения;
- позволява създаването на надеждни и сигурни приложения, дори когато изходния код е с много голям размер;
- много добра поддръжка за работа в мрежова среда - позволява създаването на приложения както от страна на клиента, така и от страна на сървъра;
- възможност за създаване на разпределени мрежови приложения;
- опростен и по-разбираем синтаксис при сравнение с C++.

3.1. Платформена независимост на Java

Един език за програмиране е платформено независим, ако позволява създаването на програмни модули, които без да се прекомпилират могат да бъдат стартирани на разнотипни апаратни платформи. За да се гарантира платформената независимост на Java за всяка апаратна платформа се разработва отделна JVM, която има за задача да интерпретира Java Byte Code (JBC), който се получава след компилиране на изходния Java код. Интерпретирането се свежда до преобразуване на JBC до изпълним код за процесора от дадена апаратна платформа. Това разбира се води до забавяне на изпълнението на приложението, но печалбата си струва: едно Java приложение се разработва еднократно, а след това, без да се налага прекомпилирането му, може да се използва при разнотипни апаратни платформи, както е показано на фиг.1.3. Това е особено важно за приложения, които функционират в Интернет среда.



Фиг. 1.3. Платформена независимост на Java

3.2. Ключови думи в Java

Ключовите думи са запазени и се използват от компилатора. Използването на ключова дума с цел деклариране на променлива, метод или клас е забранено. В Табл. 1.1 е направена класификация на по-голяма част от ключовите думи в Java.

Табл. 1.1. Класификация на ключовите думи в Java

Типове данни
byte, short, int, long, float, double, char, boolean, void
Булеви стойности
true, false
Модификатори на достъпа
abstract, protected, private, public, final, static
Обработка на изключителни събития (грешки)
try, catch, finally, throw, throws
Наследяване на клас и реализация на интерфейс
extends, implements
Управление на изпълнението
if-else, do-while, for, break, continue, switch, case
Получаване и проверка на инстанция на обекти
this, instanceof
Защита на споделени ресурси
synchronized
Други
class, new, null, import, package

3.3. Категории данни в Java

Както всеки език за програмиране, така и Java поддържа определен набор *примитивни* типове данни. Тяхната класификация по категории е показана в Табл. 1.2.

Табл. 1.2. Категории и типове примитивни данни в Java

Категория	Тип на данните
Целочислени	byte, short, int, long
С плаваща запетая	float, double
Символи	char
Булеви	boolean

Размерността, минималната и максимална стойност за всеки примитивен тип данни са дадени в Табл. 1.3.

Табл. 1.3. Размерност на различните типове данни в Java

Тип на данните	Размерност, бита	Минимална стойност	Максимална стойност
byte	8	-128	+127
short	16	-2^{15}	$+2^{15}-1$
int	32	-2^{31}	$+2^{31}-1$
long	64	-2^{63}	$+2^{63}-1$
float	32	IEEE 754	IEEE 754

double	64	IEEE 754	IEEE 754
char	16	0	2^{16}-1
boolean	1 (true / false)	-	-

3.4. Java оператори

Операторите в Java до голяма степен съвпадат по име и предназначение с операторите на C++. Те приемат един или повече аргументи и като резултат връщат нова стойност. Следва класификация на по-важните оператори:

3.4.1. Математически оператори

Това са едни от най-често използваните оператори: събиране (**+**), изваждане (-), умножение(*****), деление(**/**) и остатък от делението (**%**). Особеното при Java е, че оператор (+) се използва и като оператор за слепване на низове.

3.4.2. Оператори за увеличаване и намаляване на стойност с 1

Към тази група спадат операторът за инкрементиране (**++**) и операторът за декрементиране (**--**). Java позволява предварително (++x, --y) и пост (x++, y--) инкрементиране и декрементиране.

3.4.3. Релационни оператори

Всички оператори от тази група връщат булев резултат: истина (**true**) или лъжа (**false**): по-голямо (**>**), по-малко (**<**), по-голямо или равно (**>=**), по-малко или равно (**<=**), равно (**==**), не равно (**!=**). Особеното при Java е, че когато се използват с *обекти*, оператори "==" и "!=" сравняват техните *референции*, а не *съдържанието* им. За да предотвратите евентуални проблеми, когато се налага да се сравнява съдържание, трябва да се използва метод equals.

3.4.4. Логически оператори

Операторите от тази група винаги връщат булев резултат. Те са: логическо И (**&&**), логическо ИЛИ (**||**), логическо отрицание (**!**). Най-честото им приложение е за задаване на условието при конструкциите за управление на изпълнението.

3.4.5. Оператори за побитова работа

Тази група оператори реализират основните логически операции на ниво бит за един или два операнда: И (**&**), ИЛИ (**|**), сума по модул 2 (**^**) и отрицание (**~**).

3.4.6. Оператори за побитово изместване

Тази група оператори работят с примитивни данни от тип int. Могат да се използват и за char, byte и short, но в този случай описаните типове вътрешно се преобразуват до int преди да се реализира преместването. Преместването на определен брой битове е знаково (запазва се знака на входната данна) надясно (**>>**) или наляво (**<<**). За разлика от C++ при Java се поддържа и беззнаково преместване на дясно (**>>>**).

Едно често приложение на операторите за преместване е при реализацията на бързо умножение (<<) и деление (>>>) по степените на 2, например:

```
int x2 = x1 << 3;  // умножение по 8 (2^3).
```

3.4.7. Други оператори

В тази група ще причислим по-често използваните оператори, които не са описани до този момент:

* *Троен оператор*. Синтаксисът е:

(булев израз) **?** (стойност при истина) **:** (стойност при лъжа);

Ако булевият израз има стойност true се връща стойността на израза след "?", а в противен случай - стойността на израза след ":", например:

```
int x2 = (x1 > 25) ? 100 : (x1+100);
```

Ако x1= 11, то стойността на x2 ще бъде 111;

* *Оператори за преобразуване на типа*. Java позволява явно преобразуване на типа на всички примитивни типове данни. Изключение правят булевите. За да промените типа, в скоби преди името на променливата задавате името на новия тип, например:

```
int x1 = 1000;
long x2 = (long)x1;
```

Трябва да имате предвид, че Java няма да ви позволи преобразуване, което води до загуба на информация.

* *Оператор за присвояване*. Това е операторът (**=**). Java поддържа и *съставно присвояване* с цел опростяване на записа на математически и логически изрази, например:

```
int x1 += x2;  // този запис е еквивалентен на:
int x1 = x1 + x2;
```

Освен за събиране (+), този синтаксис може да се използва и с оператори: изваждане (-), умножение (*), деление (/), побитово умножение (&), побитово ИЛИ (|) и побитова сума по модул 2 (^).

3.4.8. Приоритет на операторите

Операторите във всеки език за програмиране имат приоритет. Той трябва да се знае, когато програмистът не използва скоби (), за да укаже изрично на компилатора какъв приоритет желае. В Java приоритетът е следният (в низходящ ред): оператори за присвояване, троен оператор, логически и побитови, релационни, аритметични и за изместване, за инкрементиране и декрементиране.

3.5. Java конструкции за управление на изпълнението

Конструкциите за управление на изпълнението са аналогични на тези от C++. За разлика от C++ при Java няма оператор goto.

3.5.1. Конструкция if-else

Когато желаете да разклоните по условие вашият код най-често ще използвате тази конструкция. Нейният синтаксис е следния:

```
if (булев израз) {
    // този код се изпълнява, ако булевият израз връща истина
} else {
    // този код се изпълнява, ако булевият израз връща лъжа
}
```

Може да използвате if и без else, а може да използвате и колкото са необходими "else if" конструкции.

3.5.2. Итерационна конструкция for

Това е най-често използваната конструкция за реализация на програмни цикли. Синтаксисът е следния:

```
for (инициализация; условие за край; стъпка) {
    // тяло на цикъла
}
```

В полето за *инициализация* се задават началните стойности на променливи, от които зависи изходът от цикъла. Инициализацията се реализира преди първа итерация.
Условието за край трябва да връща булев резултат. При лъжа се излиза от цикъла. Стойността на това поле се проверява преди всяка итерация. *Стъпката* позволява промяна на стойността на променливи в края на всяка итерация. Java позволява всяко едно от тези полета да бъде оставено празно. Можете да използвате множество променливи в полетата за инициализация и стъпка, стига те да са от един тип. За разделяне на променливите се използва оператор "**,**". Това прави конструкцията for мощно средство за реализация на условни и безусловни цикли.

```
for (int i=0, j=i+5; i<4; i++, j=i*4)  {
    System.out.println("Step "+ (i+1) + ": i="+ i + " ,j=" + j);
}
```

Резултатът, след изпълнение на този код, е следния:

```
Step 1: i=0, j= 5
Step 2: i=1, j= 4
Step 3: i=2, j= 8
Step 4: i=3, j= 12
```

3.5.3. Итерационна конструкция while

Синтаксисът на конструкция while е следния:

```
while (условие) {
    // тяло на цикъла - изпълнява се докато условието има стойност истина
}
```

Чрез while(true) може да се реализира безкраен цикъл.

3.5.4. Итерационна конструкция do-while

Синтаксисът на конструкция do-while е следния:

```
do  {
    // тяло на цикъла - изпълнява се докато условието има стойност истина
} while (условие);
```

Разликата между while и do-while е в това, че при do-while тялото на цикъла се изпълнява поне един път, независимо от началната стойност на условието.

При Java 8 може да реализирате цикли чрез метод forEach. Той се използва с обекти колекции и ламбда изрази. Конструкция forEach скрива от програмиста как точно се реализира итерационната конструкция и позволява на програмиста да се фокусира върху желаната логика.

3.5.5. Конструкции break и continue

Когато се налага да се реализира изход от тялото на цикъл реализиран чрез for, while и do-while се използва break. Ако искате да преустановите само текущата итерация, а не да излезете от цикъла, трябва да използвате continue:

```
int N=100, M=50;
for(int i=0; i<N; i++)  {
    if(i==M) {
       continue;
```

```
    }
    // код, който се изпълнява, докато условието за край връща истина
}
```

3.5.6. Конструкции за избор switch

Когато желаете да използвате по-сложни логически разклонения в своите приложения вместо if-else можете да използвате конструкция switch-case. Синтаксисът е следния:

```
switch (израз) {
    case  стойност1:
        // тяло на case за стойност1
        break;
    ..............................
    case  стойностN:
        // тяло на case за стойностN
        break;
    default:
        // този код се изпълнява, когато стойността на поле "израз" не
        // съвпада с нито една от описаните чрез case стойности.
}
```

Трябва да запомните, че изразът след switch и стойностите след case могат да бъдат от тип int или char.

3.5.7. Връщане от метод - return

Всеки метод може да връща стойност - стойността на примитивна променлива или обект. Това се реализира чрез return. Ако типът на връщаната стойност, зададена в декларацията на метода, е void, return се използва само за връщане от метод.

IV. ВЪПРОСИ И ЗАДАЧИ ЗА ИЗПЪЛНЕНИЕ

1. Как се постига апаратно независим на Java програмният код? Използвайте източници в Интернет, за да анализирате как функционира Java виртуалната машина (JVM). Кои други програмни езици използват JVM или идеите, заложени при JVM? Анализирайте програмни езици C#, Kotlin и Java Android.

2. С какви примитивни типове данни работи Java. Сравнете в таблица примитивните типове, които използва Java и C#. Опишете основните разлики. Анализирайте възможността за деклариране на числа със знак и без знак при двата езика.

3. Има ли разлика между операторите на Java и C#? Опишете в таблица тези разлики.

4. Напишете програмния код чрез който се проверява дали в един масив, който съдържа цели числа, има числа, които са в интервал [99, 125] и са нечетни. Да се разпечата на конзолата индекса и стойността на всички числа, които отговарят на тези условия.

В Java, както при много други програмни езици, масиви се декларират чрез използване на квадратни скоби [ ]. Когато се налага чрез един израз да се декларира и инициализира масив, то синтаксисът е следният:

```
int array[] = {23, 45, 99, 120, 123, 340, 500};
```

Типът на елементите в масива е int. Както при останалите езици за програмиране, така и при Java в един масив може да има само *еднотипни* елементи. Индексирането на елементите на масива започва от 0. Броят на елементите на един масив се получават лесно, тъй като Java поддържа за всеки масив вътрешно поле `length`, което съдържа

броя на елементите на масива. В конкретният случай може да получите броя на елементите на масива чрез следния израз:

```
int N = array.length;
```

За да визуализирате на конзолата низ може да използвате метод `println`, например:

```
System.out.println("Низ");
```

В Java оператор + може да се използва с цел слепване на низове или стойността на променливи с низове, например:

```
String новНиз = "Низ 1" + "Низ 2";
String резултат = "Низ" + variable1;
```

За да проверите дали едно число е четно или нечетно използвайте оператор %. Едно число е четно, ако резултатът е 0, в противен случай – 1. Проверката дали числото, записано в променлива `digit` е нечетно, е следната:

```
digit % 2 == 1
```

За да тествате вашето приложение инсталирайте развойната среда с която ще работите. Вижте съдържанието на файл

Какво_да_инсталитам.pdf

Създайте нов проект – Java конзолно приложение (виж Фиг. 1.4). Средата ще генерира структурата на проекта, който ще съдържа един клас в тялото на който е метод `main`. В тялото на метод `main` трябва да вмъкнете вашия програмен код.

```
package example_1;

/**
 *
 * @author usersx
 */
public class Example_1 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // Вмъкнете тук вашия код
    }

}
```

Фиг. 1.4. „Празен” Java проект (конзолно приложение)

Ако имате проблем с решението на задачата, разгледайте програмния код, показан на Фиг. 1.5. При изпълнение на този код на конзолата се визуализира следния резултат:

```
Index: 2, Digit: 99
Index: 4, Digit: 123
```

```
int array[] = {23, 45, 99, 120, 123, 340, 500};
for (int index=0; index<array.length; index++) {
    int digit = array[index];
    if (digit % 2 == 1 && digit >= 99 && digit <= 125)
        System.out.println("Index: " + index + ", Digit: " + digit);
}
```

Фиг. 1.5. Примерно „решение” на задачата

Това е коректен резултат. Проблемът е, че този програмен код е изключително *некачествен*. Трябва да се стремите програмният код, който пишете, да е лесно модифицируем и надежден. Основните „проблеми” на програмния код от Фиг. 1.5 са следните:

- Програмният цикъл извлича броя на елементите на масива при всяко преминаване през тялото му. Броят на елементите на масива трябва да се дефинира като променлива, извън тялото на `for`.
- Конструкция `if` няма тяло – фигурни скоби { }. В случая това е допустимо, тъй като в тялото на `if` има само един израз, който печата низ на конзолата. Ако обаче в бъдеще се наложи да се разшири програмния код от тялото на `if`, програмистът може да се забрави скобите!
- Не е добра идея да обедините всички 3 условия за елементите на масива в едно. По добрият вариант е първо да проверите дали числото е в зададения интервал и само тогава да проверите дали е нечетно.
- Никога не задавайте в кода конкретни стойности на числа. В случая променлива `digit` се проверява дали е в указания интервал, като границите му са зададени като числа. Ако в бъдеще се наложи границите на интервала да се променят, това ще стане трудно, тъй като трябва да се търси къде в кода участват тези числа (те може да участват на много места, а не само на едно, както е в случая). Винаги използвайте константи, за да декларирате тези числа. В Java константа се декларира чрез ключова дума `final`.
- Печатът на конзолата може да се подобри, като се въведе някакво форматиране.

На Фиг. 1.6 е показан коригирания програмен код.

```
final int MIN_NUMBER = 99;
final int MAX_NUMBER = 125;
final int ODD = 1;
final int EVEN = 0;

int N = array.length;

for (int index=0; index<N; index++) {
    int digit = array[index];
    if (digit >= MIN_NUMBER && digit <= MAX_NUMBER) {
        if (digit % 2 == ODD) {
            String result =
                    String.format("Index: %d, Digit: %d",index, digit);
            System.out.println(result);
        }
    }
}
```

Фиг. 1.6. Коригиран програмен код

5. Една крава изяжда тревата от 1m^2 площ с постоянна скорост от n минути. Изчислете за колко време (във формат брой дни, часове и минути) ще са необходими на кравата, за да изяде тревата на стадион с размери w x h метра. Имайте предвид, че кравата яде, без да почива. Всички входни данни (n, w и h) са цели числа.

Проверка на решението: При n = 9, w = 104 и h = 71 трябва да получите следния резултат:

```
Време за изяждане на тревата на стадиона: 46 дни, 3 часа и 36 минути
```

Задачата ще бъде решена на Семинарно упражнение №1.